ENCICLOPEDIA DEL BASIC

LeG

SPECTRUM 1

ABC de la programación



ceac

SPECTRUM 1

ABC de la programación



SPECTRUM 1

ABC de la programación

ediciones ceac

Perú, 164 - 08020 Barcelona - España

Primera edición: Enero 1987

ISBN 84-329-7711-X (Rústica)
ISBN 84-329-7716-0 (Rústica especial)
ISBN 84-329-7721-7 (Cartoné)
ISBN 84-329-7728-4 (Cartoné especial)
ISBN 84-329-7726-8 (Cartoné, Obra completa)
ISBN 84-329-7727-6 (Cartoné, Obra completa especial)

Depósito Legal: B-44037 - 1986

Impreso por
GERSA, Industria Gráfica
Tambor del Bruc, 6
08970 Sant Joan Despí (Barcelona)

Printed in Spain
Impreso en España

Contenido

Presentación

Esta Enciclopedia del BASIC, que ahora comienza, le va a introducir en uno de los temas más apasionantes y actuales de nuestros días: la programación de los ordenadores. Al mismo tiempo, al aprender el lenguaje de programación BASIC, como lenguaje diseñado especialmente para dar los primeros pasos en programación, está sentando las bases para el estudio de cualquier otro lenguaje de alto nivel.

Tal vez su preocupación en este momento, sea la de saber qué podrá hacer cuando acabe. Pues bien, el grado de habilidad en programación en lenguaje BASIC, que usted alcanzará y cómo lo alcanzará, lo podemos ilustrar con el ejemplo del proceso seguido al aprender su propio idioma:

— Primero aprendió a reconocer letras y sílabas y a escribirlas en un papel, sin establecer todavía ninguna conexión entre ellas.

— Después, pasó a establecer la conexión entre las sílabas hasta conseguir la habilidad suficiente, tanto en lectura como en escritura, para formar frases y textos relativamente largos.

— Por último, llegó a comprender cualquier texto que leía, a establecer la conexión de la lectura con la experiencia del mundo que le rodea y a expresar por escrito sus propias ideas y su relación con el mundo circundante.

Un mecanismo semejante es el que seguirá al estudiar la programación en lenguaje BASIC.

— Una primera fase será necesaria para reconocer las acciones elementales que realiza el ordenador y comprender los efectos que desencadena una instrucción dada al ordenador al pulsar una tecla.

— Una segunda fase servirá para aprender a combinar las acciones elementales, realizando así los primeros programas.

— Finalmente, una tercera fase le permitirá comprender ya la conexión de los programas con el problema que pretende resolver. Es decir, será capaz de leer y comprender programas escritos por otros y será capaz de escribir usted mismo los programas para abordar los problemas que pretende resolver.

Permítame, antes de acabar esta presentación de la Enciclopedia, que haga dos observaciones importantes:

— La primera, es que no debe confundirse la habilidad para escribir un programa y la habilidad para resolver un problema. Por ejemplo, para escribir un programa que resuelva ecuaciones de segundo grado, además de saber programar, se requiere saber resolver ecuaciones. Por eso, usted podrá programar aquellos problemas y campos que conozca.

— La segunda, es referente al estilo de programar. Nosotros le enseñaremos a escribir correctamente. Pero, lo mismo que un escritor se distingue por su estilo e imaginación, un programador debe formar y cultivar un estilo, de manera que le lleve a razonamientos lógicos y precisos, y una imaginación para ver los problemas desde ópticas muy variadas.

Ahora sólo quiero decirle que, todos los que hemos colaborado en la elaboración de esta Enciclopedia lo hemos hecho con la única preocupación de que usted consiga su objetivo, y por eso, nuestro esfuerzo se ha orientado a hacer su estudio fácil, agradable y práctico.

Josep Maria Riera
Director de la Enciclopedia

Aclaración importante

Observará el lector de esta «Enciclopedia del BASIC» que la numeración de capítulos es correlativa, iniciándose en el Tomo I de la misma y finalizando en el Tomo V y último.

Hemos creído conveniente tal continuidad, en primer lugar por el carácter secuencial de la Enciclopedia, al ser los temas que la constituyen correlativos entre sí y formando un todo inseparable; y en segundo lugar porque estimamos que ello facilita la *rápida localización de cualquier tema,* lo que, al tratarse de una obra de consulta frecuente, supone una indudable ventaja para el lector.

Composición y estructura de la Enciclopedia

La Enciclopedia está dividida en 5 tomos, cada uno de los cuales está compuesto por dos partes. Una, que constituye el texto principal y otra, que lleva el subtítulo de «Prácticas con el microordenador».

Las dos forman un todo unitario y deben ser estudiadas en paralelo. En el texto principal estudiará las características del BASIC estándar, mientras que en las «Prácticas con el microordenador» estudiará las características específicas del microordenador con el que usted trabaja.

En cuanto a la estructura de la Enciclopedia, en 5 tomos aunque interrelacionados, constituyen un peldaño o nivel sobre el que usted va avanzando hacia el objetivo final:

— En el primer tomo «ABC de la Programación» (Capítulos 1 a 4) estudiará todas las instrucciones básicas para elaborar un programa, la entrada de datos, la manipulación de memoria y la salida de datos de forma adaptada al interlocutor humano.

A estos elementos se añaden los relativos al cálculo, de modo que pueda utilizar el ordenador como una potente máquina de calcular. En este sentido, al final de este nivel, y si sus conocimientos matemáticos se lo permiten, estará en condiciones de realizar cálculos complicados, aunque no domine todavía el control de alternativas, tan importante en el tratamiento de la información. En todo caso, si sus conocimientos matemáticos no son tan altos, siempre podrá trabajar con aquellas operaciones que conoce.

— El segundo tomo «Estructura de la Programación» (Capítulos 5 a 8) introduce todas las bases para la composición condicional; es decir, para

saber tratar aquellos casos en que se presentan diferentes alternativas: «Si ocurre tal cosa, hacer esto; si ocurre tal otra, hacer lo otro...»

En este tomo se introduce la lógica binaria para hacer correctamente las preguntas al ordenador, la bifurcación como instrucción elemental para tomar decisiones y todos los operadores booleanos (no se preocupe si alguna de estas palabras no le suena mucho, pues las asimilará sin dificultad).

Tambien aprenderá la forma de almacenar programas en la cassette y el modo de cargar programas previamente grabados. Se termina con los ordinogramas, que le permitirán diseñar programas, independientemente del lenguaje que después vaya a utilizar.

En este segundo tomo se encuentra la esencia de la programación y la mayoría de las instrucciones y aspectos, que se estudian, son válidos para cualquier lenguaje de programación.

— El tercer tomo, «Cómo simplificar y mejorar los programas» (Capítulos 9 a 12) estudia las estructuras y composición de la repetición. Los bucles, que son la repetición de las mismas instrucciones para muchos elementos iguales, tienen especial aplicación en el tratamiento de casos repetitivos; por ejemplo, la confección de facturas y recibos, y son una herramiento básica de programación. De hecho, se trata de casos particulares de la composición condicional y secuencial, vistas anteriormente. Dada la frecuencia de estos casos, todos los lenguajes de programación, y el BASIC no es una excepción, tienen instrucciones especiales para tratarlos.

Estudiará también la confección de tablas y matrices, indispensables en innumerables aplicaciones.

— El cuarto volumen, «La lógica del ordenador. Gráficos y Sonido» (Capítulos 13 a 15), comprende una serie de instrucciones relativas a la estructura de la programación; por su importancia destaca la instrucción de ir a una subrutina, que es el elemento más utilizado para descomponer un programa en trozos cortos y comprender mejor el problema en su conjunto. Es, además, una herramienta fundamental cuando el programa se realiza en grupo y cada uno de los componentes lleva a cabo una parte del mismo.
También se estudia la forma de realizar toda clase de gráficos y dibujos en color, así como la obtención de sonido

— Finalmente, en el quinto tomo «Más allá del Basic» (Capítulos 16 a 18), se hace una introducción a temas más avanzados como el conocimiento interno de la máquina, los distintos sistemas para almacenar información, la organización de ficheros, el lenguaje máquina, otros lenguajes de programación distintos del Basic.

Asimismo, se estudian diversos programas de aplicación como los procesadores de textos, la hoja electrónica y las bases de datos.

Cómo estudiar en esta Enciclopedia

Al comenzar cada Capítulo encontrará un *esquema de contenido*. La finalidad de este esquema es doble:

— Por una parte, le ofrece una visión panorámica de todos los temas que va a estudiar en ese capítulo.

— Por otra, si posteriormente debe repasar algún punto determinado, le facilitará el poder localizarlo.

Leálos despacio para tener esta visión panorámica.

Después del esquema de contenido, cada capítulo comienza definiendo sus objetivos. De esta manera usted sabrá desde el principio lo que aprenderá en él. También le servirá como referencia para saber si el objetivo marcado ha sido alcanzado por usted.

A lo largo de los capítulos y al final de los mismos encontrará unos *resúmenes*, seguidos de unos *ejercicios de autocomprobación*. Su finalidad es hacer un alto en el camino y recapitular lo estudiado desde la última parada. Al mismo tiempo, los ejercicios de autocomprobación le servirán para que usted mismo compruebe si ha asimilado los conceptos estudiados y si está en disposición de continuar adelante o, por el contrario, si es necesario volver a repasar algo antes de seguir. La solución a los ejercicios de autocomprobación la encontrará al final de la primera parte del volumen.

Le recomendamos también que, cuando deba interrumpir su estudio, lo haga siempre al final de un capítulo o al terminar de resolver alguno de los grupos de ejercicios de autocomprobación que aparecen a lo largo de las lecciones.

Al hablar de la composición de la Enciclopedia, le dijimos que cada volumen se componía de dos partes. La del texto principal o estudio del BASIC estándar, y la de «Práctica con el microordenador», que viene a ser un complemento de la anterior. Ahora que va a comenzar a estudiarlo comprenderá enseguida la razón de esta comparación.

Cuando se aprende un idioma es frecuente que uno se encuentre con la sorpresa de que en determinadas regiones aquello que uno aprendió a decirlo de una manera se diga de otra. Con el lenguaje de programación BASIC pasa lo mismo. Cada fabricante introduce en el uso de su ordenador un dialecto distinto, que normalmente coincidirá en su mayor parte con el

BASIC estándar, pero tendrá suficientes características especiales como para que el que comienza se encuentre ante su ordenador sin saber qué hacer en determinados momentos.

Por eso, como nuestra intención es que aprenda BASIC y que lo practique con *su microordenador* hemos utilizado la Enciclopedia de modo que usted no se encuentre perdido en ningún momento. Así, al estudiar la lección del BASIC estándar es como si dijéramos: *Esto se dice así normalmente en lenguaje BASIC*. Sin embargo, cuando haya diferencias, añadiremos en el capítulo «Prácticas con el microordenador»: *Ojo, que esto mismo en su microordenador se dice de esta forma*. De este modo, podrá dialogar tranquilamente con su microordenador sin desagradables sorpresas y traducir al lenguaje que entiende su microordenador un programa que en BASIC estándar pueda estar escrito de forma un poco diferente.

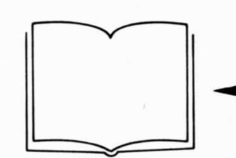
Concretamente, en el estudio de las dos partes debe proceder del modo siguiente:

— Comience siempre cada capítulo por el texto principal y continúe hasta que encuentre esta viñeta en el margen izquierdo:



Verá que en la pantalla de la viñeta aparecen unos números. Concretamente 1.1. Esto indica que debe dejar en este momento el capítulo que está estudiando, y pasar al apartado 1.1. de «Prácticas con el microordenador».

— Continúe ahora el estudio de «Prácticas con el microordenador» hasta que encuentre esta otra viñeta:



En ella se le remite de nuevo al capítulo que dejó anteriormente, para continuar donde lo interrumpió. También aquí se le indica el sitio exacto donde debe continuar. En todo caso, le recomendamos que deje siempre una señal en la página donde interrumpe el estudio. Así tendrá siempre la página localizada.

Observará el lector de esta «Enciclopedia del BASIC» que la numeración de capítulos es correlativa, iniciándose en el Tomo I de la misma y finalizando en el Tomo V y último.

Hemos creído conveniente tal continuidad, en primer lugar por el carácter secuencial de la Enciclopedia, al ser los temas que la constituyen correlativos entre sí y formando un todo inseparable; y en segundo lugar porque estimamos que ello facilita la *rápida localización de cualquier tema*, lo que, al tratarse de una obra de consulta frecuente, supone una indudable ventaja para el lector.

Parte I
BASIC

Capítulo 1
• Programación de ordenadores.
• Primeras instrucciones del BASIC.

ESQUEMA DE CONTENIDO

Objetivos

La programación y los ordenadores
- ¿Qué es un programa?
- Análisis de las instrucciones
- Automatización del proceso
- El ordenador

Evolución de los ordenadores

Arquitectura del microordenador
- El ordenador
  - Unidad Central de Procesos
  - Memoria primaria o principal
- Dispositivos periféricos
  - Teclado
  - Pantalla de televisión
  - Impresora
  - Cassette
- Memoria secundaria
  - Disco magnético
- El funcionamiento básico del ordenador

El software
- Los lenguajes de programación
- El lenguaje BASIC

Las primeras instrucciones de BASIC
- Instrucción PRINT
- Constantes
- Variables
- Instrucción LET

Los programas BASIC
- Sentencia
- El Programa
- Adición de una sentencia
- Modificación de una sentencia

Instrucción INPUT

1.0 OBJETIVOS

Dos son los objetivos que debe alcanzar después del estudio de este primer capítulo:

Primero: Entender el concepto de programa.

Segundo: Saber de qué elementos se compone el ordenador.

El concepto de programa se estudia dentro del ámbito del lenguaje de programación BASIC, que es el que estudiaremos en la Enciclopedia.

La numeración de las instrucciones y el modo de escribirlas son las ideas básicas que hay que entender para determinar el funcionamiento de un programa.

Debe aprender a clasificar los datos en numéricos y textuales, pues son estos datos los que determinan qué hace un programa.

Aprender lo que es una variable y para qué sirve es una necesidad primordial, pues toda la programación se basa en este concepto. Debe poner mucha atención en distinguir el contenido de la variable y el nombre de la variable; ésta es la mayor dificultad que tiene el capítulo.

Las tres instrucciones PRINT, LET e INPUT son la base de todos los programas y es difícil imaginar un programa que no las contenga.

La instrucción PRINT es la que nos permite inspeccionar lo que hay dentro del ordenador.

La instrucción LET nos sirve para manipular el contenido de la memoria.

La instrucción INPUT nos sirve para introducir datos al ordenador.

De estas instrucciones sólo se estudian en estos momentos algunos aspectos básicos para comenzar a trabajar. Ya las veremos posteriormente más a fondo.

Finalmente, en el aspecto de programación se estudian los comandos u órdenes, que son las herramientas necesarias para hacer funcionar el ordenador con los programas que escribimos.

La descripción de los elementos del ordenador es otro aspecto que se estudia en este capítulo. Se introduce el vocabulario básico para nombrar las distintas partes del ordenador, su organización y su funcionamiento.

El punto más importante, desde el punto de vista de la programación, son los conceptos de bit y octeto que están muy relacionados con los programas.

1.1 LA PROGRAMACION Y LOS ORDENADORES

1.1.1 ¿Qué es un programa?

La primera idea de programa la podemos tomar de la vida cotidiana. La explicación de cómo se cambia una bombilla, cómo se hace una tortilla, cómo se abre una puerta, o bien cuando utilizamos unas instrucciones para montar un avión de juguete, para manejar un radio-cassette, o conectar un

electrodoméstico; o sea, *el proceso de aplicar unas instrucciones es ejecutar un programa; el proceso de escribirlas es hacer un programa.*

Definición de programa

> Definimos un programa como una serie de instrucciones que hay que aplicar en un orden estricto para conseguir un objetivo más complicado.

El objetivo de esta Enciclopedia es aprender a programar una máquina, es decir, saber establecer la secuencia de instrucciones que la máquina realizará para llegar al fin que hemos establecido previamente.

La figura 1 muestra la factura de una papelería. En ella se distinguen claramente dos tipos de letra, una de imprenta y otra escrita a mano. La

SUMINISTROS DE PAPELERIA, S. A.
Paseo de la Luna, 23
08017 BARCELONA

Barcelona, 1/4/85

COLEGIO MGC
c/. Artistas, 27
08026 Barcelona

ARTICULO	CANTIDAD	PRECIO UNITARIO	IMPORTE
Paquetes de 100 folios	3	150	450
Lápices nº 2	5	20	100
Gomas de borrar	10	5	50
		Total	600
	IVA	12%	72
			672

Figura 1. Modelo de factura.

letra de tipo imprenta y los recuadros definen el papel que se denomina factura en blanco. La escrita a mano corresponde al resultado de rellenar la factura.

La tarea que nos proponemos realizar es establecer unas instrucciones para rellenar la factura. Insistimos en que el objetivo no es escribir la factura, sino plantear las instrucciones para rellenar cualquier factura, o sea, realizar el programa que servirá para hacer facturas.

Instrucciones

Unas instrucciones para escribir la factura las tenemos en la figura 2, que vamos a comentar seguidamente. Vaya comparando los pasos con el contenido de la factura.

Los pasos 1 y 2 nos indican cómo hay que escribir la fecha de la factura y los datos del cliente.

PASO 1. Escribir en parte superior derecha el lugar y la fecha de la factura.
PASO 2. Escribir en el interior del recuadro superior derecha el nombre y la dirección del cliente.
PASO 3. Escribir la línea para un artículo.
  PASO 3.1. Escribir el nombre del artículo debajo de la columna de artículo.
  PASO 3.2. Escribir en la columna de cantidad el número de unidades que ha comprado el cliente.
  PASO 3.3. Escribir en la columna de precio unitario el valor de una unidad del artículo después de consultar una lista de precios.
  PASO 3.4. Calcular el valor de la compra del artículo multiplicando el precio unitario por la cantidad.
  PASO 3.5. Escribir en la columna de importe la cantidad obtenida.
  PASO 3.6. Si quedan más artículos repetir los pasos 3.1 a 3.5.
PASO 4. Escribir la raya de separación debajo de la columna importe.
PASO 5. Sumar la columna de importe.
PASO 6. Escribir «Total» y el resultado de la suma debajo de la columna de importe.
PASO 7. Calcular el impuesto multiplicando el total por 3 y dividiendo por 100.
PASO 8. Escribir el valor del impuesto debajo de la columna de importe.
PASO 9. Escribir la raya de separación debajo de la columna de importe.
PASO 10. Escribir «Total» y el resultado de añadir al total anterior el valor del impuesto.
PASO 11. Escribir una raya doble debajo de la columna importe.

Figura 2. Instrucciones para la confección de una factura

El paso 3 son las instrucciones para escribir la línea correspondiente a un artículo. Este paso se repite tantas veces como artículos distintos ha comprado el cliente.

Finalmente en los pasos 4 al 11 describimos cómo realizar los totales de la factura y el cálculo del impuesto correspondiente al valor de la factura.

Observemos que la instrucción 3 se desglosa en los subpasos del 3.1 al 3.6. Esto es necesario para expresar una instrucción que es demasiado complicada para escribirla en una línea.

En este caso se supone que se dispone de una lista de precios que no aparece ni en la factura ni en las instrucciones.

1.1.2 Análisis de las instrucciones

Las instrucciones para realizar la factura poseen dos propiedades importantes:

Primera propiedad: Cada instrucción lleva un número para definir un orden estricto de realización.

Orden de ejecución

El paso 7, por ejemplo, debe realizarse antes que el 8 y después del 6 para asegurar un procedimiento correcto en la elaboración de la factura.

Estructura de la instrucción

Segunda propiedad: Cada instrucción se compone de un verbo, que indica la acción que se desea realizar y unos complementos que indican el qué, dónde, etc.

Por ejemplo, la instrucción 4 se compone del verbo «escribir» para indicar la acción y el complemento que indica lo que debe hacer. En este caso la raya de separación.

La instrucción 9 es igual que la 4. La instrucción 8, aunque mantiene la misma acción de «escribir», escribe otra cosa: el «valor del impuesto».

En la instrucción 2 además de la acción, «escribir», del qué, el «nombre y dirección del cliente», indica dónde la colocamos, «en el interior del recuadro superior derecha».

En la instrucción 3.4, la acción es distinta, es «calcular». El resto de la instrucción indica qué calculamos (el «valor del artículo»), la operación «multiplicando» y qué valores debemos multiplicar (el «precio unitario» y la «cantidad»). Esta instrucción se puede expresar de forma equivalente de la manera siguiente: «Multiplicar el precio unitario por la cantidad para obtener el valor del artículo».

Sintaxis y significado

La conclusión más importante que sacamos del ejemplo es la mezcla de una parte de gramática y de una parte de significado. Las instrucciones individuales deben estar escritas con reglas gramaticales estrictas y a la vez hay que elegir las palabras con un significado preciso.

Consideremos la instrucción número 6; la frase escribir «Total» no queda clara, pues no se dice el lugar exacto donde debemos escribir «Total». La interpretación de la instrucción es probablemente correcta porque se supone que la persona (el sujeto) tiene ya conocimientos previos.

En el caso de los ordenadores el lenguaje utilizado en la descripción del proceso es mucho más rígido que el lenguaje corriente, debido a que el ordenador no posee estos conocimientos previos. Por lo tanto, al darle

la instrucción de escribir «Total», le deberíamos indicar también dónde escribirla.

> En resumen, la técnica de escribir un programa consiste en descomponer el problema en pasos elementales; es decir, que utilicen un solo verbo, y numerarlos adecuadamente para establecer un orden estricto de ejecución.

1.1.3 Automatización del proceso. El ordenador

En el ejemplo considerado en el apartado anterior suponíamos que el sujeto que realizaba las acciones era una persona humana ayudada con lápiz y papel.

Estudiemos las posibilidades de automatizar el proceso para rellenar (hacer) una factura.

La primera idea que se nos ocurre es dotar al que realiza la factura de una máquina de calcular y una máquina de escribir.

El control del proceso

Mejoramos en dos aspectos: el primero es una mayor rapidez y seguridad en el cálculo, y el segundo una mejor presentación de la escritura. Sin embargo, el aumento de rapidez en el proceso global no es demasiado apreciable. La razón es la necesidad de teclear en la calculadora y en la máquina de escribir los resultados intermedios y los nombres, al ritmo del sujeto humano.

Todo el control general del proceso lo realiza el cerebro de la persona (sujeto) que lo ejecuta y las herramientas adicionales mejoran la calidad del resultado, pero no aportan una mejora sustancial en la velocidad.

La utilización del ordenador, en cambio, mantiene los requisitos de calidad con un incremento considerable de velocidad.

Características que aporta el ordenador

¿Qué características ha añadido el ordenador respecto a las herramientas tradicionales: máquina de escribir y máquina de calcular? La respuesta es que el *ordenador posee memoria y capacidad de decisión* para ejercer el control sobre el proceso global.

El ordenador es capaz de recordar los resultados intermedios y, lo que es más importante, las instrucciones para realizar la factura. El ordenador es capaz de aprender a ejercer el control y tomar las decisiones oportunas para la realización de la factura.

En la realización manual, la memoria se localiza en la persona que ejecuta las instrucciones, aporta todos los elementos necesarios para que el lápiz, el papel, la máquina de escribir y la máquina de calcular cooperen para lograr un objetivo útil.

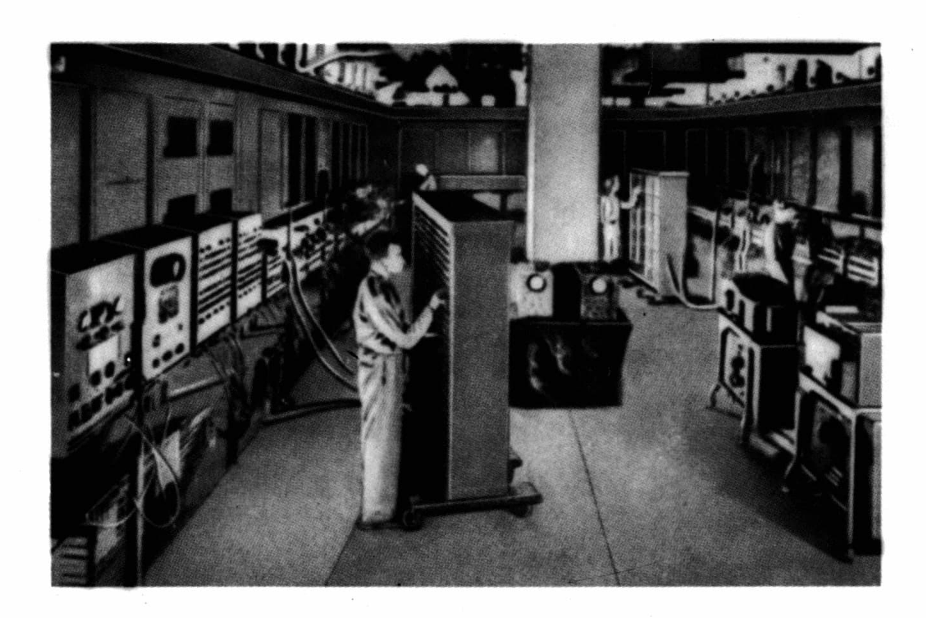
La memoria del ordenador nos permite almacenar toda la información necesaria para realizar una tarea previamente diseñada. La labor que emprendemos consiste en aprender a colocar en la memoria de la máquina las instrucciones precisas para que el trabajo se realice correctamente.

1.2 EVOLUCION DE LOS ORDENADORES

Factores de la evolución

La historia del progreso de los ordenadores, no es más que la historia de la tecnología para construir memoria.

Figura 3. Panorámica general del primer ordenador ENIAC



Los factores que marcan saltos importantes en esta evolución son:

— El proceso de miniaturización. Conseguir más memoria en menos espacio.

— La velocidad de acceso. Es la velocidad con que el ordenador puede leer y escribir en la memoria.

Repasemos brevemente cómo ha ocurrido el desarrollo de este proceso.

Tecnología de válvulas

El primer ordenador lo construyeron en la Universidad de Pennsilvania en el año 1947 los profesores Echert y Mauchly. Lo bautizaron con el nombre de ENIAC (Electronic Numerical Integrator and Calculator) que significa calculador e integrador numérico electrónico. La tecnología básica de su funcionamiento eran circuitos electrónicos a válvulas. Disponía de 1000 memorias aproximadamente, que correspondían a 18000 válvulas. Las dimensiones físicas eran de unos 20 metros de largo por 3 de alto y su peso era de 30 toneladas. El dato más curioso es que el tiempo promedio entre dos averías del ENIAC era de 30 minutos. La figura 3 nos muestra el ENIAC.

El 14 de junio de 1951 se presenta en el mercado norteamericano el primer ordenador fabricado en serie, con tecnología de válvulas electrónicas. Se inicia lo que se ha llamado la *Primera Generación de Ordenadores,* es decir, todos los ordenadores cuyos componentes electrónicos son válvulas.

Tecnología de transistores

En 1960 se sustituyen las válvulas por transistores, elementos que realizan las mismas funciones que las válvulas pero son mucho más pequeños. Compare en la figura 4 el tamaño de una válvula con el tamaño de un transistor. Las memorias se transforman en pequeños núcleos de hierro mag-

Figura 4. *Componentes electrónicos de la Primera y Segunda Generación de ordenadores: a) Válvula electrónica. b) Transistor*



netizable denominados núcleos de ferrita. Se inicia la *Segunda Generación de Ordenadores,* el número de memorias crece hasta un valor de 16000 y las dimensiones del ordenador se reducen a 1 metro de largo, ancho y alto. El tiempo entre averías se mide ahora en meses.

Tecnología de circuitos integrados

La *Tercera Generación de Ordenadores* se inicia en 1965 cuando los transistores son sustituidos por circuitos integrados. Circuitos completos de miles de transistores se colocan en una pastilla de silicio de 1 centímetro cuadrado. El silicio es un material que se encuentra en abundante proporción en la arena de las playas. Las mismas funciones se realizan en circuitos de tamaño mucho más pequeño. A los ordenadores construidos con estos circuitos se les llama miniordenadores. Los miniordenadores realizan las mismas operaciones que los ordenadores pero el tamaño se ha reducido sensiblemente.

A principio de la década de los 70 se descubren los sustitutos de los núcleos de ferrita, son las *memorias electrónicas* construidas sobre circuitos integrados que permiten abaratar mucho el coste de la memoria.

Microprocesador

En 1974 se construye el primer *microprocesador;* es un circuito integrado, que contiene una unidad completa para realizar todo el cálculo que necesita un ordenador. Recuerde que sus dimensiones son del orden de 1 centímetro cuadrado.

La introducción del microprocesador supone un paso decisivo para que la Informática se ponga al alcance de cualquier presupuesto, ya que el componente que era más caro del ordenador pasa a ser prácticamente el más barato.

Como muestra la figura 5, ahora un ordenador puede caber en un cuadrado de 20 centímetros de lado con gran holgura, se utiliza como herramienta de cálculo el microprocesador y como memoria, las memorias elec-

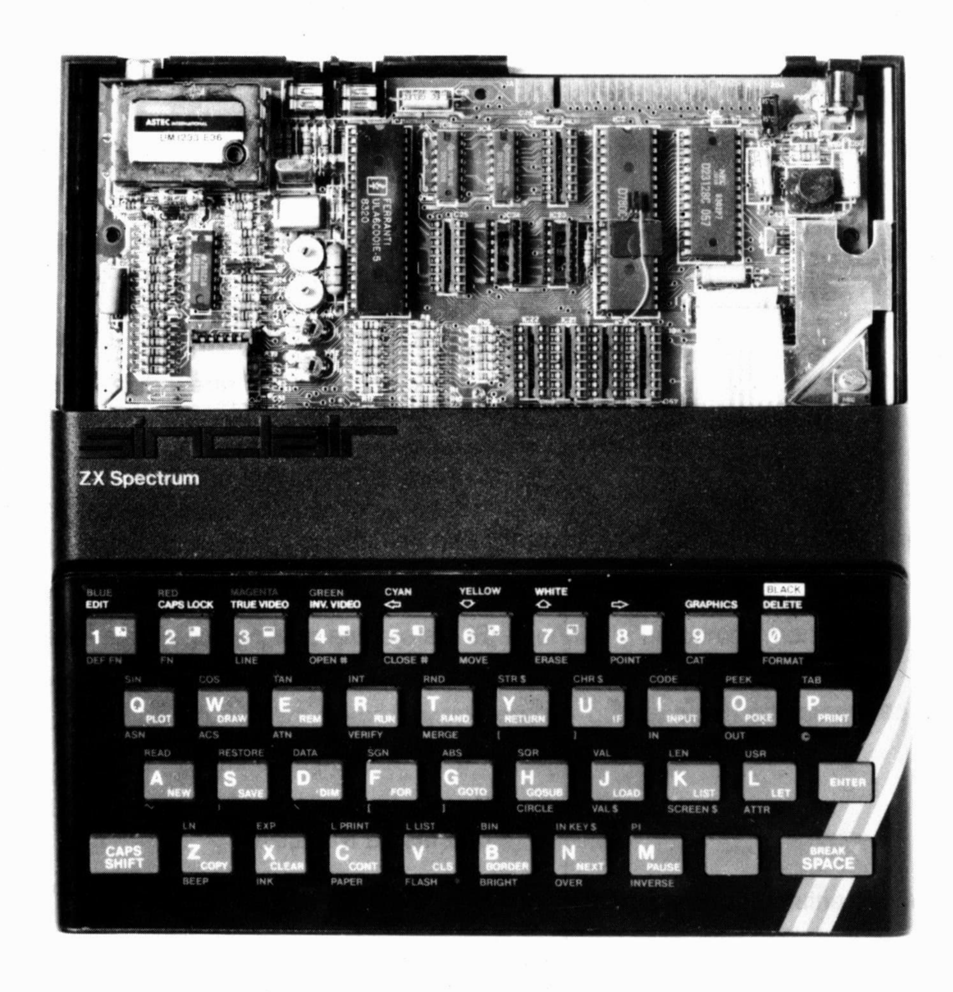


Figura 5. Circuitos integrados de un microordenador

trónicas. Debido a su pequeño tamaño a un ordenador construido con un microprocesador y con memorias electrónicas se denomina *microordenador.*

Hay que señalar que los términos ordenador, miniordenador y microordenador describen objetos que realizan las mismas funciones, admiten programas iguales y su comportamiento es idéntico siempre que el número de memorias y la velocidad de ejecución de las instrucciones sea la misma.

Ordenador, miniordenador y microordenador

La diferencia de ordenador, miniordenador y microordenador únicamente se refiere al tamaño total del objeto. Sin embargo, puede decirse con bastante generalidad que los ordenadores tienen más memoria y mayor velocidad que sus hermanos los miniordenadores y también que la memoria y la velocidad del miniordenador es mayor que la de un microordenador.

Si comparamos el ordenador a otros inventos modernos típicos nos daremos cuenta del avance real que supone en el quehacer humano.

La máquina de vapor multiplica por 100 la fuerza que realiza el hombre.

Un coche multiplica por 10 y un avión por 100 nuestra capacidad de transporte.

Un ordenador aumenta nuestra capacidad de cálculo en un millón de veces.

La elocuencia de los números habla por sí misma.

RESUMEN

Un programa es una serie de instrucciones que hay que realizar en un orden estricto.

Cada instrucción se numera para fijar el orden de realización.

Cada instrucción se compone de:

— Un verbo que indica la acción a realizar.

— Unos complementos que indican el qué, dónde y cómo.

La persona o el ordenador son el sujeto que realiza las acciones y no aparece en las instrucciones.

La característica principal del sujeto es que tiene memoria para recordar las instrucciones a aplicar y capacidad de decisión.

Estas instrucciones componen un lenguaje:

— Si las ejecuta una persona sirve el lenguaje corriente.

— Si las ejecuta una máquina es necesario un lenguaje más rígido.

Escribir un programa consiste en:

— Descomponer un problema en instrucciones adecuadas al sujeto que las va a realizar.

— Numerarlas convenientemente para definir su orden de ejecución.

El ordenador es un sujeto adecuado para ejecutar programas por que tiene memoria y capacidad de decisión.

Los ordenadores han evolucionado a base de reducir el tamaño de sus memorias y de aumentar la velocidad de acceso a la misma.

Esquemáticamente la evolución de los ordenadores se resume en:

GENERACION	**TECNOLOGIA**
PRIMERA	Válvulas electrónicas
SEGUNDA	Transistor y Núcleos de Ferrita.
TERCERA	Circuito Integrado Microprocesador Memorias Electrónicas

El tamaño define el uso de los términos ordenador, miniordenador y microordenador.

EJERCICIOS DE AUTOCOMPROBACION

Encierre en un círculo la letra que corresponde a la alternativa correcta:

1. Hacer un programa consiste en:

   a) Calcular una factura
   b) Definir lo que hay que hacer
   c) Enunciar las instrucciones y especificar su orden de ejecución.
   d) Especificar unas instrucciones para escribir una sola factura.

2. La redacción de instrucciones correcta debe contener los elementos siguientes:

   a) Un complemento que indique dónde y un complemento que indique el qué
   b) Un número de orden, un verbo, y quizás algún complemento.
   c) Cuándo hay que realizar la acción
   d) Un número de orden y qué hay que hacer

3. Una propiedad que diferencia un ordenador y una máquina de calcular es:

   a) La velocidad de cálculo
   b) Que el ordenador realiza operaciones que la máquina de calcular no puede hacer
   c) Que el ordenador calcula muchas operaciones de golpe y la máquina de calcular no
   d) Que el ordenador memoriza las instrucciones que debe realizar

1.3 ARQUITECTURA DEL MICROORDENADOR

La figura 6 muestra un esquema de las partes principales del ordenador. Hay que distinguir dos partes:

— El ordenador propiamente dicho, parte tramada de la figura 6, y

— Los dispositivos periféricos, lo que está contenido en la parte clara de la figura.

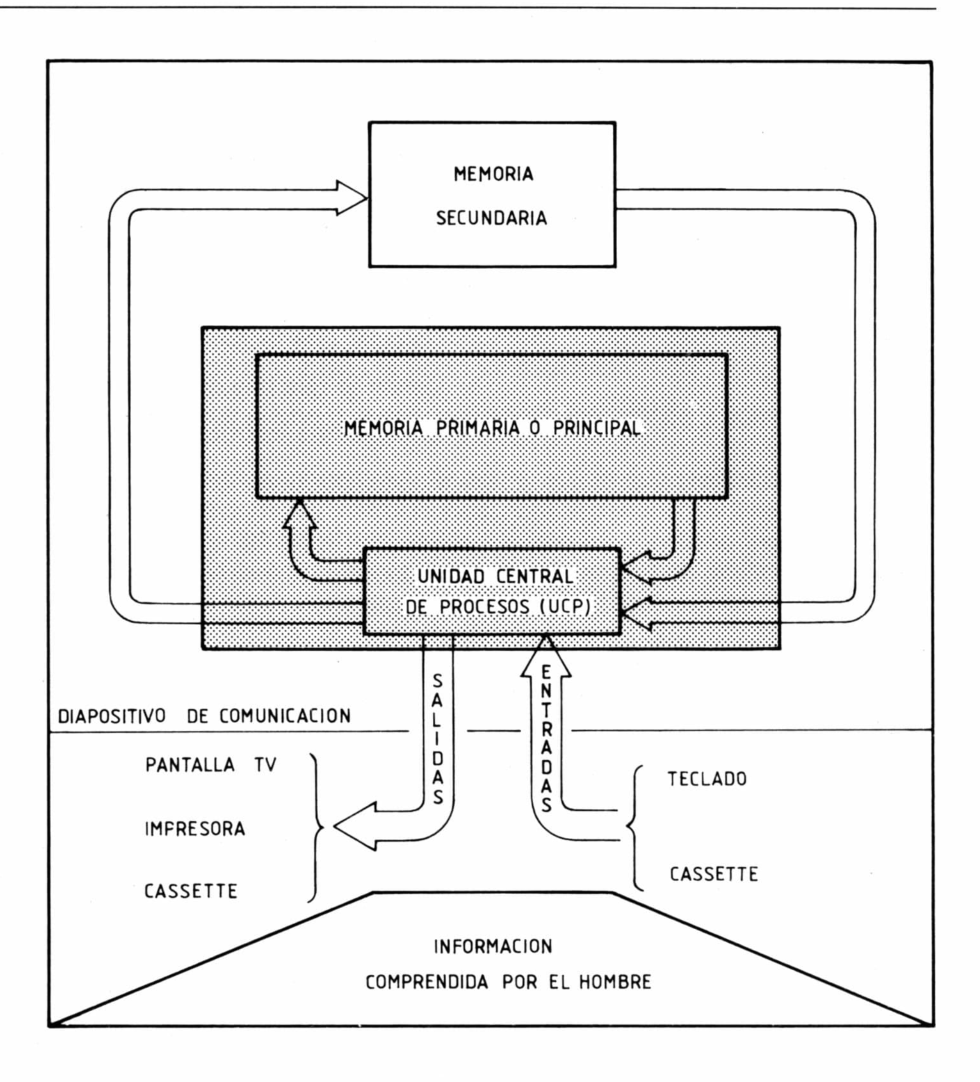


Figura 6. Partes de un microordenador

Partes del ordenador

A su vez el ordenador se divide en dos partes:

— Unidad central de procesos.
— Memoria principal o memoria primaria.

Los dispositivos periféricos se clasifican en dos tipos:

— Dispositivos de comunicación.
— Memoria secundaria.

A continuación se describe cada uno de ellos.

1.3.1 El ordenador

Está compuesto por:

1.3.1.1 Unidad Central de Procesos

La abreviatura de la Unidad Central de Proceso es UCP y nos referiremos siempre a ella mediante estas siglas. Debemos advertir que en muchos libros y manuales esta parte se abrevia como CPU, del nombre inglés Central Processing Unit.

Componentes de la UCP

La UCP se compone al menos de un microprocesador que es el que efectúa las operaciones y un vibrador o reloj que marca el ritmo a que se efectúan las operaciones.

En un microordenador sólo se dispone de un microprocesador en la CPU, mientras que en los ordenadores y miniordenadores esta unidad se compone en la actualidad de varios microprocesadores que tienen tareas especializadas y también de un reloj para marcar el ritmo de realización de las operaciones.

La constitución de la UCP es la diferencia más grande, aparte del tamaño, entre un ordenador, un miniordenador y un microordenador. Un microordenador lleva únicamente un microprocesador que realiza todas las tareas de cálculo sin ningún tipo de especialización; en cambio, los ordenadores y miniordenadores llevan más de un microprocesador.

Misión de la UCP

La misión de la UCP es realizar todos los cálculos que el ordenador necesita (la máquina de calcular), y regular todas las lecturas y escrituras en la memoria y en los dispositivos periféricos.

Como se indica en la figura 6, la UCP está conectada al resto de elementos mediante cables con más o menos hilos eléctricos, que en la figura 6 se simbolizan por flechas.

1.3.1.2 Memoria primaria o principal

Es el conjunto de circuitos que memorizan los datos y las instrucciones. Cada circuito elemental se denomina báscula electrónica y se caracteriza por tener sólo 2 posiciones posibles.

Esta combinación de dos posiciones (o también posibilidades) define la mínima información que memoriza el ordenador y se denomina *bit.*

El bit, elemento más pequeño de la memoria

El bit no es sinónimo de báscula electrónica, es un término de significado más general. Todos los dispositivos que son capaces de mantener dos posibilidades representan el bit. Unos ejemplos aclaran este concepto: Un interruptor de una bombilla representa un bit porque sólo puede estar encendido y apagado. Un núcleo de hierro (ferrita) puede estar magnetizado o no.

La memoria, por lo tanto, consiste en un elevado número de dispositivos que representan bits. Como norma general, para distinguir en qué estado se encuentra el dispositivo, se utilizan los números 0 y 1 que representan los valores que puede tener el bit. La asociación concreta del número 0 a un estado del dispositivo (y en consecuencia el 1 para el estado contrario) es una cuestión que deciden los fabricantes de los ordenadores, y depende de cada modelo. Por lo tanto, si tomamos el ejemplo del interruptor, el 1 no

justifica necesariamente encendido y el 0 apagado, pues podría ser también lo contrario. Esto dependerá, como decimos, del fabricante.

Este tipo de detalles no son importantes, pues todos los fabricantes respetan las propiedades aritméticas y lógicas de los valores 0 y 1 que necesitamos para la programación.

Celdas del bit

La memoria aunque consta de bits, desde el punto de vista del programador se divide en grupos de más o menos bits. La figura 7 muestra esta subdivisión de la memoria en grupos o celdas de varios bits. Tomando el ejemplo de las bombillas, cada bombilla representa un bit (y puede estar encendida o apagada como muestra la figura); y cada celda es un grupo de varias bombillas. Así la primera fila de la figura constituirá una celda de 8 bits.

Estos agrupamientos se hacen por dos razones:

Mayor posibilidad de información

Primera razón: En la práctica se manipulan *informaciones que representan más de dos posiciones.* Para contemplarlas hay que utilizar grupos de bits. Si tomamos dos bombillas, podemos representar cuatro posibilidades:

Posibilidad 1: Las dos bombillas apagadas

Posibilidad 2: Primera bombilla apagada y la segunda encendida

Posibilidad 3: La primera bombilla encendida y la segunda apagada

Posibilidad 4: Las dos bombillas encendidas.

Sustituyendo las bombillas por dos bits se representan las cuatro posibilidades por las combinaciones 00, 01, 10, 11, el primer número repre-

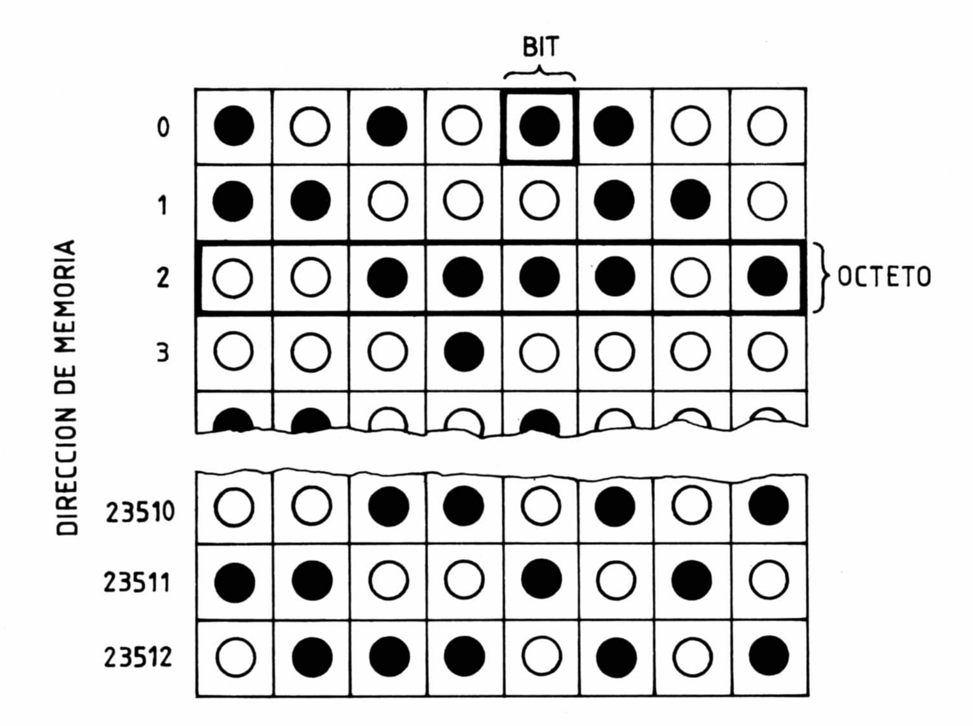


Figura 7. Organización de la memoria de un ordenador

senta el valor del primer bit y el segundo representa el valor del segundo bit, etc. Con tres bit representamos 8 posibilidades 000, 001, 010, 011, 100, 101, 110, 111.

Aumento de velocidad de transferencia

Segunda razón: Se gana *velocidad de transferencia o de acceso a la memoria.* Las operaciones de lectura y escritura manipulan varios bits de golpe. Por ejemplo, si tenemos que trasladar 5 libros de una estantería a otra, podemos hacerlo cogiendo un libro cada vez, o bien si disponemos de una carretilla trasladarlos todos de golpe. Para poder elegir la segunda opción hay que disponer de la carretilla. El tamaño de la carretilla es importante, si hay que trasladar 50 libros y en la carretilla sólo caben 5, deberemos realizar 10 viajes. En el ordenador hay circuitos electrónicos que realizan las funciones de la carretilla para transportar bits.

Se han ensayado agrupamientos de 8, 9, 12, 16, 24, 32 bits. El número de 8 bits se ha consolidado como número idóneo en los ordenadores pues es una buena solución entre la velocidad de transferencia y la sencillez de los circuitos. Por otra parte, la información que contienen 8 bits es de un total de 256 posiciones o posibilidades, que es un número más que suficiente para describir, por ejemplo, todas las teclas de la máquina de escribir.

Byte y Kilobyte

La asociación de los 8 bits se denomina *octeto,* aunque también es muy popular la palabra inglesa *byte* (pronúnciese bait). La memoria queda dividida en celdas de 8 bits y todas las operaciones de transferencia mueven por lo menos un octeto.

El tamaño de la memoria se mide en kilobytes y se abrevia KB. Un kilobyte representa 1024 bytes u octetos de memoria. Así, un ordenador con 16 KB de memoria significa que dispone de 16384 (=16 por 1024) celdas de grupos de 8 bits. Es importante observar que a diferencia de la utilización normal de la palabra kilo que significa 1000, en la medida de la memoria se utiliza la palabra kilo significando 1024.

La figura 7 muestra esta división en octetos; cada bit se representa por un cuadrado con un círculo en el centro, puede imaginarse como una bombilla. Los puntos negros significan que la bombilla está apagada, o lo que es lo mismo que el valor del bit es cero. Los puntos blancos representan la bombilla encendida, por lo tanto, el valor del bit es 1. Cada fila de bits consta de 8 bits que definen el octeto. Cada fila representa, por lo tanto, una celda de memoria.

Dirección de memoria

Cada celda de la memoria lleva asignada un número que se denomina dirección de memoria. Así la primera celda es la dirección 0, la segunda es la dirección 1 y así sucesivamente. Esta dirección la utiliza la UCP para indicar de qué celda quiere leer o escribir. Observe en la figura 7 cómo cada fila, que corresponde a una celda o a un octeto, lleva un número adosado a la izquierda que corresponde a la dirección de memoria mencionada.

En resumen, la memoria del ordenador consiste en una hilera de celdas de 8 bits cada una, que se denomina octeto o byte. Van numeradas desde el cero hasta el número de bytes que tiene la memoria. Al número asignado a cada byte se denomina dirección de memoria.

1.3.2 Dispositivos periféricos

La UCP y la memoria constituyen propiamente el ordenador. Sin embargo, por sí mismo, el ordenador no hace absolutamente nada si no recibe datos y programas del exterior. Los ingleses utilizan para describir esta situación la palabra hardware que significa chatarra, quincalla, máquina inútil.

Comunicación del ordenador

Por otra parte, el interior del ordenador sólo reacciona a impulsos eléctricos. Por eso, para comunicar al ordenador los programas y datos necesarios, precisamos algún dispositivo que transforme nuestras órdenes en señales eléctricas.

De la misma manera,el ordenador que elabora los resultados en forma de impulsos eléctricos, necesita disponer de algún dispositivo que transforme las señales eléctricas en formas que comprendamos.

Al conjunto de dispositivos que permiten la comunicación del exterior con el ordenador y del ordenador con el exterior se denominan periféricos.

Se disponen de gran variedad de periféricos. Nos limitaremos, por el momento, a describir brevemente los más importantes.

1.3.2.1 Teclado

Es un periférico especializado en la comunicación desde el hombre al ordenador. Se clasifica como periférico de entrada.

Como se observa en la figura 8, consiste en unas teclas semejantes a las de una máquina de escribir. La disposición es parecida a las mecanográficas, aunque suele tener unas teclas especiales de las que no dispone la máquina de escribir.

Cuando se aprieta una tecla los circuitos electrónicos de este periférico transforman la pulsación en una señal eléctrica capaz de memorizarse en el ordenador.

El teclado es diferente para cada ordenador. En líneas generales todos los teclados son parecidos, pero las diferencias son lo suficientemente im-



Figura 8. Teclado de un ordenador

portantes para obligar a un aprendizaje al usuario del ordenador antes de poder trabajar con él.

En las «Prácticas con el microordenador» que se refieren a un modelo de máquina concreto, encontrará información detallada de su teclado. Ya le indicaremos en qué momento debe estudiarla.

1.3.2.2 Pantalla de televisión

Es un periférico especializado en la comunicación del microordenador al hombre. Se clasifica como periférico de salida.

Consiste en un circuito electrónico que transforma los impulsos eléctricos en los dibujos de los símbolos que aparecen en la pantalla de televisión (letras, cifras, símbolos especiales, etc.).

1.3.2.3 Impresora

Es un periférico especializado en la comunicación del microordenador al hombre. Se clasifica como periférico de salida.

Consiste en un circuito electrónico que transforma los impulsos eléctricos en pulsaciones automáticas de una máquina de escribir.

1.3.2.4 Cassette

Es un periférico tanto de entrada como de salida. Su misión consiste en guardar la información en forma de sonido, no precisamente musical sino más bien desagradable.

Memoria permanente

La utilidad de este periférico es almacenar de manera permanente datos y programas. Al apagar la corriente del ordenador todo lo que tenemos en memoria desaparece, al volver a conectarlo a la corriente la memoria aparecerá limpia de todo rastro de programas y datos anteriores. Para recargar un programa deberíamos volver a teclear los programas cada vez que conectamos el ordenador a la corriente. La cinta cassette permite cargar esta información sin nuestra intervención.

Intercambio de información

Una utilidad adicional consiste en permitir un intercambio de datos y programas entre varios ordenadores. El contenido de la memoria de un ordenador se copia en una cinta y entonces esta información puede leerse en el otro ordenador.

1.3.3 Memoria secundaria

La memoria primaria o principal es muy rápida en velocidad de acceso, pero es relativamente limitada en capacidad. Por otra parte, muchas veces sólo se desea trabajar con una parte limitada de datos. Mientras hacemos la factura del cliente A no la hacemos del cliente B, sólo se precisan en este momento los datos del cliente A.

Características

La memoria secundaria tiene las mismas características que la memoria principal, pero la transferencia de datos es unas 10000 veces más lenta, aunque la capacidad es unas 100 veces mayor, siendo esta cifra muy variable.

Organización

En general, la organización es semejante a la de la memoria principal pero las celdas son mayores que las de la memoria principal. Estas celdas dependen mucho del dispositivo que se utilice.

Memoria permanente

Una diferencia muy importante de la memoria secundaria respecto a la memoria principal es que la memoria secundaria es permanente. Cuando se interrumpe el flujo de corriente los datos quedan intactos (no se descargan, no desaparecen, se conservan).

El dispositivo físico por excelencia que se utiliza como memoria secundaria es el disco magnético.

1.3.3.1 Disco magnético

Consiste en una capa de material magnético depositada sobre un disco aproximadamente del mismo tamaño que un disco microsurco sencillo. El mecanismo de memorizar consiste en magnetizar o no magnetizar una porción de material.

La utilización más común en programación es mantener los ficheros en la memoria secundaria. De esta manera cuando se consulta una ficha sólo se coloca en la memoria principal la información relativa a la misma, sin necesidad de mantener todo el fichero en la memoria principal. Este sistema permite disponer de ficheros mucho mayores que la capacidad de la memoria principal.

1.3.4 El funcionamiento básico del ordenador

Ya hemos mencionado que la figura 6 indica además de la organización del ordenador algunos aspectos de su funcionamiento. Las flechas simbolizan las autopistas por donde viajan los datos.

Transferencia de datos

Si observamos la dirección de las flechas vemos que van desde la UCP a las memorias principal o secundarias y de éstas a la UCP. Indicamos, así, que el movimiento es en las dos direcciones, la memoria transfiere datos a la UCP y viceversa.

Las flechas que van a los periféricos de comunicación, sin embargo, tienen un sentido único. Los periféricos de entrada (teclado, cassette) transmiten datos desde el periférico a la UCP. Los periféricos de salida (pantalla, impresora, cassette) reciben los datos desde la UCP.

No todos los periféricos son de sentido único, por ejemplo, la cinta del cassette aparece en la flecha de las entradas y en la flecha de las salidas.

Control del tráfico de datos

La UCP es el regulador de este tráfico. Observemos que todas las flechas llegan o parten de la UCP, no hay flechas que conecten la memoria con los periféricos. Para transferir un dato desde el teclado a la memoria se utiliza la UCP de intermedio. En otras palabras, el dato viaja desde el teclado a la UCP y desde la UCP hasta la memoria.

El ritmo de estos intercambios los marca un vibrador o reloj de tal manera que la secuencia de trabajo se distribuye en el tiempo según el reloj va desgranando impulsos.

En el intervalo de tiempo entre una vibración y la siguiente la UCP sólo realiza una operación, ya sea de cálculo o de transferencia de datos. No todas las operaciones tardan el mismo tiempo; es más larga una suma que

una lectura en memoria. Sin embargo, cuando se hace una suma y después una lectura en memoria, el tiempo total que tarda el ordenador es de dos impulsos de reloj, aunque las operaciones sean de duración distinta. Cada impulso de reloj desencadena una operación. En consecuencia, el tiempo entre dos impulsos de reloj debe ser más largo que el tiempo que tarda la UCP en realizar la operación más larga.

Ritmo de ejecución

El ritmo del reloj es frenético, realiza alrededor de unos cuatro millones de impulsos en un segundo; o lo que es lo mismo, lanza un impulso cada 250 mil millonésimas de segundo; si lo escribimos con decimales, diríamos cada 0,00000025 segundos. Debido a este ritmo muchas de las operaciones del ordenador parecen instantáneas; recuerde que la sensación de continuidad que da el cine se debe a imágenes fijas que se proyectan durante una décima de segundo.

RESUMEN

Las partes del ordenador son:

— Ordenador propiamente dicho. Se compone de:

Unidad Central de Procesos (UCP) que consiste en:

Uno o varios microprocesadores que son la máquina de calcular del ordenador y el regulador del tráfico de todos los datos.

Un vibrador o reloj que marca el ritmo a que se van realizando las instrucciones.

En la memoria primaria o principal debemos considerar estos aspectos:

Se compone de unidades de mínima información, el BIT, que sólo puede tener 2 posiciones o posibilidades, y que se representan por un 0 y un 1.

Se agrupan en celdas por dos razones: Para ganar velocidad de transferencia y para manipular informaciones con más posiciones.

El agrupamiento de 8 bits se denomina octeto o byte que permite almacenar informaciones con 256 posiciones distintas.

Cada celda se distingue de las otras por una dirección de memoria.

La medida de la memoria se expresa en kilobytes (KB). Un KB representa 1024 octetos. La palabra kilo tiene un significado distinto del que se usa en la expresión de otras medidas, pues equivale a 1024 bytes.

Cuando desconectamos el ordenador todo el contenido de la memoria principal se pierde.

Los dispositivos periféricos son complementos al ordenador que realizan tareas muy especializadas. Se dividen en dos tipos:

Dispositivos de comunicación: Sirven para transformar señales eléctricas en formas que comprendemos (letras, dígitos, etc.) y entonces son dispositivos de salida. O bien a la inversa, transforman formas que entendemos en señales eléctricas para el ordenador y entonces son dispositivos de entrada.

Los más comunes son:

Teclado: Dispositivo de Entrada.

Pantalla: Dispositivo de Salida.

Impresora: Dispositivo de Salida.

Cassette: Dispositivo de Entrada y de Salida.

Memoria secundaria: Es una memoria mayor que la principal y más lenta que la principal.

La característica más importante es que mantiene los datos cuando desconectamos el ordenador de la corriente.

El dispositivo más representativo de la memoria secundaria es el disco magnético.

EJERCICIOS DE AUTOCOMPROBACION

4. Haga una figura con las partes de un ordenador siguientes: Memoria principal, CPU, Memoria secundaria y los dispositivos periféricos clasificados según sean de entrada, o de salida o ambos.

Encierre en un círculo la letra que corresponde a la alternativa correcta:

5. La memoria de un ordenador se organiza como:

   a) Una sucesión de bits con el acceso restringido al primero de cada octeto.

   b) Una sucesión de celdas numeradas compuestas de 8 bits.

   c) Una sucesión de celdas numeradas que tienen alta velocidad de transferencia con 8 bytes cada una.

   d) Una sucesión de celdas que cada una contiene el número del byte correspondiente.

6. Una máquina de 8 KB contiene:

   a) 8000 bytes

   b) 8016 bytes

   c) 8200 bytes

   d) 8192 bytes

7. La memoria secundaria es:

   a) Un dispositivo que tiene una velocidad de transferencia de información muy rápida.

   b) Es una memoria parecida a la principal pero permanente, es decir, mantiene la información cuando se corta la electricidad.

   c) Un disco magnético en que la información se graba en forma de sonido.

   d) Es una parte de la memoria principal que se utiliza menos.

8. Coloque un círculo en ENTRADA y/o SALIDA según el periférico de que se trate.

   8.1 TECLADO: Entrada/Salida

   8.2 IMPRESORA: Entrada/Salida

   8.3 CASSETTE: Entrada/Salida

   8.4 PANTALLA TV: Entrada/Salida

1.4 EL SOFTWARE

En el apartado anterior se ha definido la palabra hardware que designa cosas que no sirven para nada.

La pregunta que nos asalta a continuación es: ¿Cómo transformar esto que no sirve para nada en algo útil?

La contestación no es fácil si fuéramos los primeros en formularla. Afortunadamente hay muchas personas que han trabajado para resolverla y han abierto un camino que para nosotros será mucho más sencillo de recorrer.

Los programas dan personalidad al ordenador

Actualmente se dispone de programas que facilitan mucho la tarea de realizar algo útil con un ordenador. En palabras inglesas y en la jerga informática el conjunto de programas de que dispone un ordenador se denomina *software.*

El término software es contrapuesto al de hardware, e indica el conjunto de elementos lógicos y conceptuales que transforman al hardware en una cosa útil. En definitiva, el software es el conjunto de programas que posee el ordenador cuando lo compramos.

Se debe advertir que la palabra española *logial* significa software. Preferimos, sin embargo, utilizar la palabra inglesa a lo largo de esta Enciclopedia pues el término logical no se ha impuesto en el vocabulario profesional de la informática.

Hay que señalar que el concepto de software es relativo. Cuando realizamos un programa con el software disponible en el ordenador, somos a la vez usuarios porque utilizamos el programa de que dispone la máquina, y productores de software porque añadimos más programas a la máquina. Todos los programas que usted desarrolla pasan a engrosar el software de su ordenador.

1.4.1 Los lenguajes de programación

Al principio del capítulo hemos resaltado de una manera muy general la importancia de la gramática y del significado para confeccionar los programas. La redacción de unas instrucciones para realizar una cosa es muy distinta a la redacción de una novela, un informe o una carta.

El lenguaje del ordenador

El trabajo de programar es un trabajo de precisión. Si la gramática de una sola instrucción es incorrecta, o bien su significado es impreciso, el programa no funciona. El lenguaje que directamente entiende el ordenador es sumamente limitado, utiliza un juego de unas 200 operaciones para definir todas las acciones posibles. Se le indica qué instrucción se quiere realizar mediante un número que la selecciona del conjunto mencionado.

Veamos un ejemplo, si queremos especificar a la máquina que realice una operación tan elemental como 2 más 2. Haremos los siguientes pasos: (No se preocupe por no saber todavía lo que significan los números que se mencionan. Sólo queremos que vea los pasos que se necesitan).

Paso 1: Leer el dos de la memoria 4087

Paso 2: Sumar en la UCP la dirección de memoria 4087

Paso 3: Escribir desde la UCP a la dirección de memoria 5067

queremos decir con esto que debemos memorizar los códigos siguientes:

Paso 1: 62 (Código de leer de memoria), 4087 (Dirección de memoria)

Paso 2: 134 (Código de sumar), 4087 (Dirección de memoria del sumando)

Paso 3: 119 (Código de escribir en memoria), 5067 (Dirección de memoria)

Dificultades de utilización

En este ejemplo se ha supuesto que sabíamos muchas cosas, dónde se alojaba el dos (en la posición 4087), dónde se quiere escribir el resultado (en la dirección 5067), etc. Más adelante volveremos a revisar estos códigos y la manera de utilizarse. De momento se han puesto de manifiesto dos grandes dificultades en utilizar este lenguaje:

Primera: Hay que descomponer el problema real en unos pasos tan elementales, es decir, que definen una acción muy limitada, que el número de códigos a entrar es muy elevado y difícil de asegurar que es correcto.

Segunda: Sólo manipulamos números, y aunque somos capaces de entenderlos somos muy lentos en interpretarlos. Es preferible manejar letras y números más cercanos al lenguaje corriente.

Programas traductores

Sin embargo, hay que señalar que el único lenguaje que entiende la máquina es el de códigos. Se ha dedicado un gran esfuerzo en construir unos programas traductores, de tal manera que las instrucciones se dan al programa de una manera más próxima al lenguaje corriente y el programa se encarga de traducirlas a los números que entiende el ordenador.

Las instrucciones que acepta un programa traductor, del tipo que hemos mencionado, definen un *lenguaje de programación.* En definitiva, un lenguaje de programación es la serie de instrucciones que acepta un programa que traduce estas instrucciones a los códigos numéricos oportunos para que el ordenador los ejecute.

Nivel del lenguaje

El grado de proximidad al lenguaje ordinario, o de otro modo, la mayor o menor expresividad del lenguaje de programación definen el nivel del lenguaje. Ciertamente, cuanto mayor nivel, mayor es la dificultad de construir el programa traductor correspondiente.

1.4.2 El lenguaje BASIC

Se ha ideado un número considerable de lenguajes para facilitar esta tarea de programación, nombres tan extraños como FORTRAN (FORmula TRANslator, traductor de fórmulas), COBOL (COmmon Bussiness Oriented Language, lenguaje orientado a los negocios), RPG (Report Program Generator, programa generador de informes), PASCAL (nombre que evoca la memoria de un físico y matemático francés), etc.

Los lenguajes de los ordenadores

En esta Enciclopedia vamos a estudiar quizás el más popular, el BASIC. Su popularidad se debe a una estructura realmente cómoda para el programador, aunque a costa de menor velocidad de ejecución.

Contrariamente a lo que sugiere (lenguaje básico), el nombre es la abreviatura de una larga expresión inglesa (Beginners All-purpose Symbolic Instruction Code), que en castellano significa Código Simbólico de Instrucciones de propósito general para principiantes.

Se desarrolló en la Universidad de Dormouth en 1963 por Kemmeny y Kurts, aunque con unas instrucciones más reducidas de las que se usan actualmente. En 1978 se define el BASIC estándar en la Asociación Americana de Estándares (BASIC ANSI-BSRX 3.60-1978).

Dialectos del BASIC

Hay que advertir que la mayoría de máquinas poseen un lenguaje BASIC que se aparta del estándar y ocasiona una verdadera legión de dialectos. De todos modos se observa un núcleo importante de instrucciones que se mantiene, del orden del 95 por ciento. El funcionamiento de las órdenes especiales o diferentes deben consultarse, además de saber utilizar las órdenes comunes, en el manual técnico de cada BASIC en particular.

Memorias ROM

Finalmente hay que señalar que el programa BASIC va incluido en la mayoría de máquinas en un tipo especial de memorias denominadas ROM (Read Only Memory, memorias de lectura únicamente) que se caracterizan por la imposibilidad de escribir en ellas. Se evita, de este modo, la destrucción del programa BASIC por una manipulación incorrecta del usuario y además mantiene la información en el momento que se desconecta el ordenador.

El tener el traductor BASIC en memorias ROM, permite en el momento de conectar, que el ordenador se coloque automáticamente en este modo de traducción.

Memorias RAM

Como contraste, las memorias que no poseen estas propiedades se denominan RAM (Random Acces Memory, memoria de acceso aleatorio).

1.5 LAS PRIMERAS INSTRUCCIONES DE BASIC

Desconectar y conectar de nuevo

Ha llegado el momento de empezar a utilizar el ordenador. En primer lugar aprenderemos a visualizar datos y a almacenar resultados intermedios. Se puede utilizar la máquina sin ningún reparo, puesto que no la dañaremos aunque le demos órdenes incorrectas. Si nos equivocamos, el ordenador simplemente nos dará un mensaje informándonos del tipo de error cometido. En el peor de los casos, si no sabemos salir del atolladero, basta con desconectar la máquina y empezar de nuevo.

1.1.

1.5.1 Instrucción PRINT

PRINT

La palabra PRINT significa «escribe» o «imprime». Se usa para indicar al ordenador que deseamos que escriba un dato en la pantalla. Se trata, pues, de una instrucción que actúa sobre una unidad de salida, en este caso la pantalla.

El cursor

Tomemos ahora el ordenador. En la pantalla observaremos que hay un símbolo en forma de pequeño recuadro o de guión (según el modelo

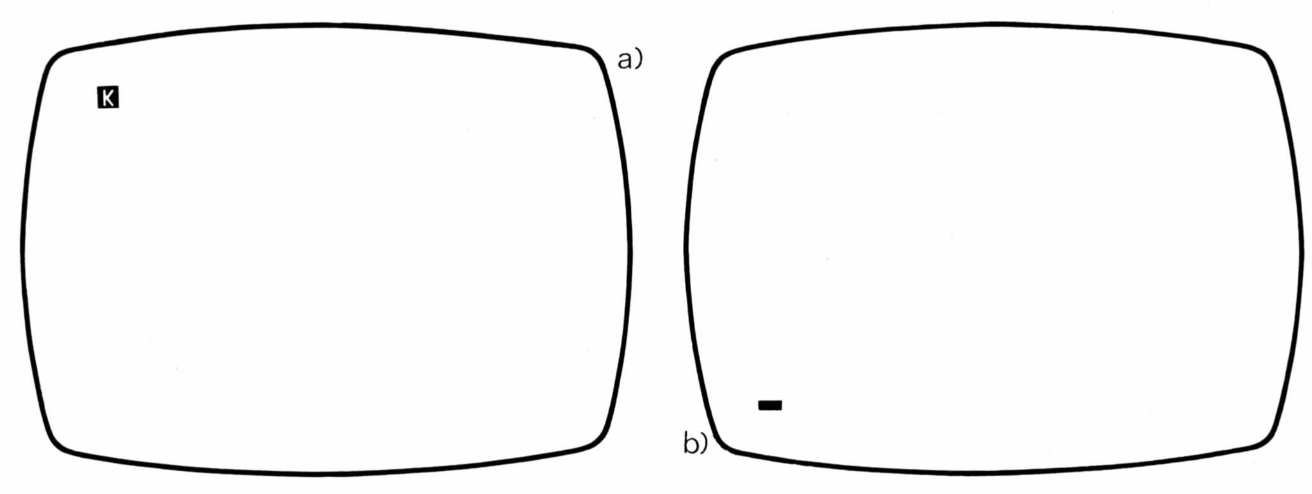


Figura 9. Cursores: a) Cursor de recuadro. b) Cursor de guión

del ordenador) que señala la posición en la cual se escribirá la siguiente letra. Este símbolo especial se denomina *cursor.* Cuando se teclean caracteres, se visualizan éstos en pantalla y el cursor se desplaza una posición hacia la derecha por cada tecla pulsada. Los dos tipos de cursores mencionados se muestran en la figura 9, en la parte (a), el cursor de recuadro y en la parte (b) el cursor de guión.

Antes de continuar, debemos hacer hincapié en un par de teclas especiales:

Tecla de retroceso

La primera de ellas es la tecla de retroceso y borrado de un símbolo escrito. La usaremos cuando se haya tecleado un símbolo equivocado y deseemos rectificarlo. Esta tecla de retroceso borra el carácter que se encuentra a la izquierda del cursor. Pulsando tantas veces como sea necesario esta tecla, rectificaremos cualquier error que hubiéramos cometido, e incluso borraremos toda la línea si hace falta. La tecla de retroceso puede estar indicada de varias formas: como BACK-SPACE (retroceso), como DELETE (borrado) o como una flecha ←. Por lo estudiado en «Prácticas con el microordenador» ya sabemos la que funciona en nuestro modelo de ordenador.

Tecla de fin de línea

La segunda tecla importante puede estar representada como ENTER (entrada), RETURN (retorno a inicio de línea) o NEW-LINE (nueva línea). Se usa para indicar al ordenador que la línea que acabamos de escribir está completa y que puede empezar a trabajar con ella. Hasta que no se pulsa esta tecla de fin de línea, el ordenador no efectúa ninguna acción y se limita a esperar la indicación de que hemos terminado de escribir la línea.

Importante: Cuando hayamos completado una línea, pulsaremos siempre a continuación la tecla de fin de línea, para indicar al ordenador que ya puede trabajar con ella. En caso contrario, la máquina se quedará esperando indefinidamente.

Puesto que ya sabemos dónde se encuentran estas dos teclas daremos nuestra primera orden a la computadora. Escribamos en pantalla la siguiente línea:

```
PRINT 3
```

Imprimir números

Esta instrucción significa «Imprime el valor 3», es decir, hemos dado la orden de que escriba un número (en este caso el 3) en la pantalla. Cuando pulsemos la tecla de fin de línea (ENTER, RETURN o NEW-LINE), se efectúa la orden y observamos que, efectivamente, aparece el número 3 escrito. ¡Hágalo! Probemos a darle una orden algo más compleja:

```
PRINT 2+3
```

Hemos dado la orden de que imprima el resultado de sumar 2 más 3. Comprobamos que al pulsar la tecla de fin de líneas, aparece escrito el valor 5.

Además de servir para escribir resultados numéricos, la instrucción PRINT también sirve para imprimir mensajes. Por ejemplo, escribamos:

```
PRINT "HOLA"
```

Imprimir textos

Al ejecutar esta orden y después de pulsar la tecla de fin de línea aparecerá en pantalla el texto HOLA. En esta instrucción se han empleado las comillas ('') para indicar al ordenador que debía imprimir el texto tal cual. Sin embargo, las comillas no formarán parte de dicho texto. Por esta razón, aparece escrito HOLA y no «HOLA». Usemos ahora una instrucccción de este tipo para indicar al ordenador que nos salude. Si escribimos.

```
PRINT "BUENOS DIAS"
```

Al pulsar la tecla de fin de línea, el ordenador escribirá BUENOS DIAS en la pantalla.

Operaciones aritméticas

En un ejemplo anterior hemos empleado la instrucción PRINT para realizar la operación de suma entre los números 2 y 3. ¿Qué otras operaciones son posibles? Aunque éste será un tema a tratar con más profundidad los capítulos posteriores, no hay inconveniente en empezar a conocer algunas de las posibilidades de que dispone el BASIC. Aprenderemos en primer lugar los símbolos que se usan para representar las operaciones aritméticas. Estos símbolos son:

+ Suma — − Resta — * Multiplicación — / División

El único símbolo un poco especial es el signo de multiplicación. En la gran mayoría de lenguajes de programación (y no sólo en BASIC) se usa el símbolo (*) denominado asterisco para indicar la multiplicación.

Para hacer un poco de práctica, escribamos las siguientes líneas para observar cómo se realizan las operaciones aritméticas. (Recordamos que debe pulsarse la tecla de fin de línea al final de cada una de ellas.)

```
PRINT 3+8+7   (en la pantalla aparecera 18)
PRINT 8-3     (en la pantalla aparecera 5)
PRINT 4*3     (en la pantalla aparecera 12)
PRINT 8/5     (en la pantalla aparecera 1.6)
```

El punto en la separación de decimales

En el último de los ejemplos, aparece un número escrito de una forma un tanto especial: las cifras decimales van separadas de las unidades por un punto en lugar de por una coma. Esto es así porque el BASIC sigue el procedimiento anglosajón para indicar los decimales (al igual que las calculadoras de bolsillo). Por tanto no sería válido escribir 1'6 ni tampoco 1,6 sino que escribiremos 1.6 utilizando el punto.

Por esta razón no deben separarse con un punto las cifras de las centenas de las cifras de los millares. Así, el número mil cien lo escribiremos 1100 ya que si escribiéramos 1.100 el BASIC lo interpretaría como el número 1 con cien milésimas.

Podríamos preguntarnos ahora cuál es el número de decimales que acepta el BASIC. Esto depende de cada máquina en concreto. Para averiguarlo escribimos una división que dé un número infinito de decimales (una fracción períodica pura). Por ejemplo 10 dividido por 3:

```
PRINT 10/3 que vale 3.333333
```

El número de cifras que contenga el resultado será el máximo de cifras que acepta nuestro ordenador. En general, para ordenadores pequeños oscilan entre 6 y 13 cifras. Averigüe las que tiene el suyo.

Sabemos ya el número máximo de cifras con las que podemos trabajar. ¿Qué ocurre si empleamos números que tengan más cifras de las admitidas? Por ejemplo, multipliquemos un millón por un millón:

```
PRINT 1000000*1000000
```

La notación exponencial

El resultado tendría 13 cifras (un 1 seguido de 12 ceros). Como este número tiene más cifras que las admitidas por un microordenador, el BASIC lo escribe de esta forma:

```
1E+12
```

Esta parece una forma un tanto extraña de escribir un número. Esta forma recibe el nombre de *notación científica* o *exponencial* y sólo la emplearemos para representar números extraordinariamente grandes o extraordinariamente pequeños. Analicemos un poco el significado de cada símbolo.

El uno de la izquierda es la cifra que estaría delante de los doce ceros. La letra E (mayúscula) significa que el número está escrito en notación exponencial. Finalmente el +12 indica que la coma decimal debe correrse 12 posiciones hacia la derecha, es decir que el número es el 1000000000000.

En realidad, el +12 significa que debemos multiplicar doce veces por 10 el número que está a la izquierda de la letra E (el 1), o lo que es lo mismo, multiplicarlo por diez elevado a 12 (de ahí proviene el nombre de notación exponencial). Así, un millón en notación científica o exponencial se escribe 1E+6. Si escribimos

```
PRINT 1E+6
```

el BASIC escribe 1000000 ya que el número de cifras está dentro del margen admitido por la máquina y, por tanto, se representa el número de forma normal. Compruébelo.

Para representar números muy pequeños los escribiremos de la misma forma, cambiando el signo + que sigue a la letra E por un signo menos. Esto significa que la coma decimal debe correrse hacia la izquierda o, lo que es lo mismo, que se dividirá por diez tantas veces como indique el número que sigue al signo menos. La instrucción

```
PRINT 1E-3
```

escribirá el valor 0.001 ya que, según lo que acabamos de decir, la coma debe correrse 3 posiciones hacia la izquierda.

Tecleemos las siguientes instrucciones para acabar de comprender estas reglas:

```
PRINT 1E+3    vale   1000 (1x10x10x10)
PRINT 1.2E+3  vale   1200 (1.2x10x10x10)
PRINT 12E+3   vale  12000 (12x10x10x10)
PRINT 1.2E-2  vale  0.012 (1.2 : (10x10)
PRINT 12E-2   vale   0.12  (12 : (10x10)
```

Como norma general, sólo emplearemos la notación científica o exponencial para representar números enormemente grandes o pequeños, pero conviene recordarla por si alguna vez es necesario emplearla.

1.5.2 Constantes

Los datos que hemos visto hasta ahora, como por ejemplo el número 3 o la palabra HOLA, se denominan CONSTANTES puesto que su valor es fijo e invariable.

Existen dos tipos de constantes: las de tipo *numérico* (por ejemplo el número 3) y las de tipo *textual* (por ejemplo la palabra HOLA). Aunque a primera vista los datos numéricos parecen los más importantes, los datos textuales (también llamados *alfanuméricos* por contener letras y dígitos) tienen mucho interés en informática. Pensemos, por ejemplo, en los nombres de las personas, en las direcciones, etc. que son informaciones de tipo textual muy utilizadas.

Los números y los textos

Para resaltar la diferencia entre los dos tipos de datos, efectuaremos las dos instrucciones siguientes:

```
PRINT 3+2
PRINT "3+2"
```

En el primer caso, como era de esperar, se escribirá en pantalla el valor 5. En cambio, en el segundo aparece 3+2 sin que se haya realizado la suma. Esto es lógico si tenemos en cuenta que la utilización de las comillas significa que se trata de un dato de tipo textual. Por tanto, aparecerá en pantalla tal como estaba escrito originalmente sin efectuar ninguna otra operación.

1.5.3 Variables

Tal como hemos utilizado hasta ahora la instrucción PRINT se escribían los resultados de unas operaciones, pero no sabemos todavía cómo almacenar resultados intermedios para utilizarlos posteriormente.

Denominación de las cajitas de memoria

Para memorizar un dato hay que definir previamente el almacén o memoria donde se va a guardar. Si definimos un almacén como el lugar donde memorizar información, podemos guardar en su interior un dato u otro según nuestra conveniencia, es decir, que su contenido no es fijo sino que, por el contrario, es variable. Precisamente por esta razón, estos almacenes o memorias se denominan variables.

Puesto que el valor de una variable es indeterminado, le daremos un nombre para referirnos a ella. Una variable sería, pues, como una caja que lleva una etiqueta (su nombre) y en su interior contiene un dato que podemos consultar o cambiar a voluntad. Esta forma de imaginarnos una variable se corresponde casi exactamente con la realidad, haciendo la salvedad que la caja es un circuito electrónico que forma parte de la memoria del ordenador. (Ver Fig. 10a.)

Variable: Espacio de memoria cuyo contenido puede variar durante la ejecución de un programa

El nombre de las variables

Toda variable debe tener un nombre que la identifique. No se puede dar cualquier nombre sino que existen unas normas de nomenclatura. Estas normas varían ligeramente de un BASIC a otro pero, en general, son las siguientes:

1) El primer símbolo del nombre debe ser una letra (de la A a la Z).

2) A continuación, si se desea, se pueden colocar letras (de la A a la Z) o dígitos (del 0 al 9).

3) Al final del nombre puede ser necesario colocar un símbolo especial para indicar el tipo de dato que contiene.

El tipo de las variables

Al definir una variable, hay que indicar también si se usará para contener datos de tipo numérico o bien de tipo textual, ya que el tamaño de almacén será distinto. El tipo de dato que contiene se indica colocando al final del nombre de la variable un símbolo predeterminado.

Para las variables de tipo textual o alfanumérico usaremos el símbolo dólar ($) como se ve en la figura 10b. Para las variables de tipo numérico no hace falta colocar ningún símbolo especial. Algunas variantes del BASIC utilizan además otros símbolos para algunos casos especiales. Sin em-

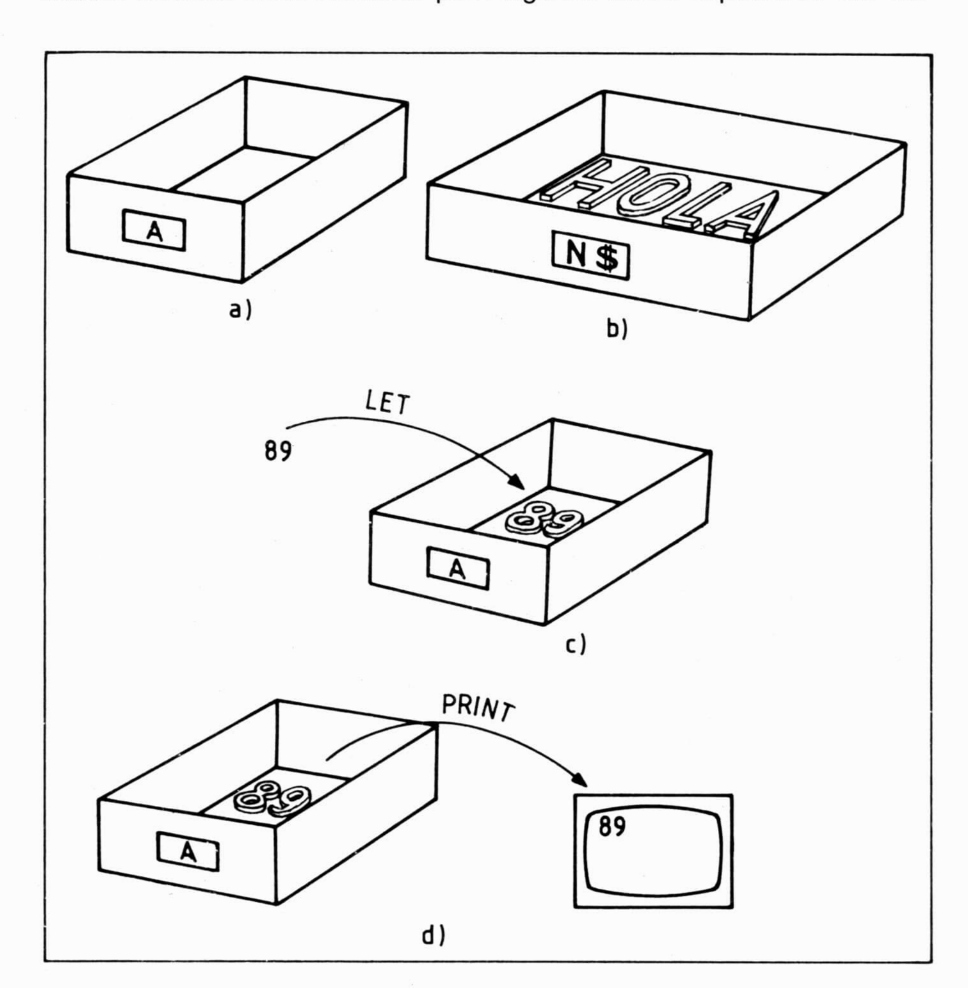


Figura 10. Manipulación de la memoria. a) Reproducción visual de la variable A. b) Variable de tipo textual. c) Memorización de un dato en una variable. d) Consulta de un dato por pantalla.

bargo, las dos reglas anteriores son válidas siempre. Veamos a continuación algunos ejemplos de nombres de variables textuales:

A$ P1$

N$ X8$

Se debe advertir que hay ciertos BASIC que sólo admiten una sola letra para variables textuales.

Seguidamente, algunos ejemplos de nombres correctos de variables numéricas:

A T

A2 TR

CL Z9

Observamos que todas ellas cumplen las tres normas enunciadas anteriormente, es decir, empiezan por una letra, a continuación pueden llevar otra letra o dígito y, si son textuales, llevan el símbolo dólar ($) al final. A continuación presentamos unos nombres de variable que no cumplen las normas y que por tanto son incorrectas:

Nombre y contenido

T-1 El guión no es un símbolo admitido para construir el nombre de una variable.

1A El nombre debe empezar por una letra.

A$2 El símbolo dólar ($) debe ir siempre al final del nombre.

> **Advertencia:** En este apartado sólo hemos descrito la manera de cómo se nombran las variables, no confunda este nombre de la variable con el contenido de la misma. Repase la figura 10, especialmente *b.*

1.5.4 Instrucción LET

LET

Hemos visto cómo definir variables para memorizar datos. Falta saber cómo realizar la operación de almacenamiento. Para ello disponemos de la instrucción LET. La forma de emplearla es la siguiente:

LET variable = valor

es decir, después de la palabra LET se escribe el nombre de la variable donde se almacenará el dato. A continuación escribimos el signo igual (=) y finalmente el valor o dato que colocaremos en la variable especificada. Por ejemplo, para almacenar el número 89 en la variable A, escribiremos

```
LET A=89
```

Asignación de valores

La palabra LET significa «sea», por tanto esta instrucción se leería «sea A igual a 89». Una vez efectuada esta instrucción, la variable A contiene el número 89. Al pulsar la tecla de fin de línea, habremos notado que la operación se ha realizado de forma «invisible», es decir, que no ha aparecido ningún mensaje escrito en la pantalla. Esto es así porque el almacenamiento se efectúa dentro de la memoria del ordenador.

Para consultar el contenido de la variable utilizaremos la instrucción PRINT. Escribiendo

```
PRINT A
```

observamos que el ordenador imprime el valor almacenado en A, que en este caso es el 89. En la figura 10c vemos un esquema del funcionamiento de la instrucción LET. Aunque en esta instrucción se emplea el signo igual (=), no tiene el significado de igualdad matemática, sino que más bien tiene el sentido de «asignación a», tal como vemos en la figura 10c. Por esta razón sería incorrecto escribir

```
LET 89=A
```

ya que la variable debe ir obligatoriamente a la izquierda del signo igual (=).

Acciones de la instrucción LET

La instrucción LET lleva a cabo dos acciones bien determinadas. La primera de ellas es crear la variable, es decir, reservar un espacio de memoria y darle el nombre elegido por nosotros. La segunda acción consiste en colocar en el interior de dicha variable un valor determinado.

Aparte de los datos numéricos, también existe la posibilidad de memorizar datos de tipo textual. Para ello emplearemos variables de tipo textual (con el signo dólar ($) al final de su nombre). Por ejemplo, para almacenar el nombre de una persona escribiremos

```
LET N$="JUAN GARCIA"
```

Recordemos que los textos van siempre entre comillas. Aquí hemos empleado la variable N$ pero hubiéramos podido utilizar cualquier otra, como por ejemplo A$ o P$. Para ver el contenido de N$ emplearemos también la instrucción PRINT, o sea

```
PRINT N$
```

El tipo de dato igual al tipo de variable

y en pantalla nos aparecerá JUAN GARCIA. Obviamente, el tipo de dato debe corresponderse con el tipo de variable, es decir, los dos numéricos o los dos textuales. Por tanto sería incorrecto escribir

```
LET A="CASA"
```

porque al tratarse de una variable textual debe llevar el signo dólar ($). También es incorrecto

```
LET X$=27
```

El valor de la variable se mantiene

porque se intenta asignar un valor numérico a una variable de tipo textual.

El valor almacenado en una variable se mantiene mientras no se indique lo contrario y puede ser consultado tantas veces como se desee. La figura 10d muestra el esquema de funcionamiento de la instrucción PRINT para el caso de escribir el contenido de una variable. Hasta ahora hemos visto cómo realizar consultas de forma simple. También es posible combinar una consulta con una operación. Por ejemplo, si escribimos

```
LET B=27
```

y a continuación tecleamos la línea

```
PRINT B
```

Consulta del valor de la variable

se imprime el número 27 en la pantalla (no olvide pulsar la tecla de fin de línea). Si ahora escribimos

```
PRINT B+3
```

en pantalla aparece el valor 30. La operación B+3 significa que tomamos el valor memorizado en la variable B (un 27) y a continuación le sumamos 3, con lo cual el resultado de la suma(30)es lo que sale escrito en la pantalla.

Sabemos que el contenido de una variable no es fijo, sino que podemos alterarlo en cualquier momento. Para ello emplearemos también la instrucción LET. En el ejemplo anterior, la variable B contenía el número 27. Si ahora escribimos

```
LET B=10
```

el valor anterior (el 27) se pierde y ahora B contiene el número 10 en su interior.

De las dos acciones que lleva a cabo la instrucción LET, a saber, crear la variable y asignarle un valor, en este caso sólo se realiza la segunda. La causa es que la variable B ya existía por haber sido creada con el LET anterior. Si ahora repetimos la instrucción

```
PRINT B+3
```

el resultado es 13, dado que B vale 10 en este momento, a la cual se le suman 3.

El valor de una variable podemos traspasarlo a otra. Por ejemplo, escribiendo

```
LET C=B
```

traspasamos el valor de B (que es 10) a la variable C. Recordemos que la variable que recibe el dato debe estar obligatoriamente a la izquierda del signo igual (=). Después de pulsar la tecla fin de línea, dele la orden

```
PRINT C
```

El resultado será 10.

El valor de la variable se copia, no se traspasa

Un hecho importante que conviene resaltar es que cuando realizamos el traspaso del valor de la variable B a C, en realidad estamos haciendo una copia de B a C. Por tanto, el valor de B no se pierde sino que tenemos dos variables, B y C, con el mismo valor.

Todas estas normas son válidas también para las variables de tipo textual, con la excepción, claro está, de que no es posible realizar operaciones aritméticas con ellas.

A la derecha del sigmo (=) de la instrucción LET hemos situado números, otras variables o textos. También se pueden escribir operaciones. Aunque insistimos en que las operaciones las veremos con más detalle en un capítulo posterior, es interesante emplearlas ahora de forma simplificada. Si escribimos

```
LET A=1+4
```

el BASIC realiza la suma de uno más cuatro y el resultado lo asigna a la variable A. Si ahora escribimos

```
LET B=2*A
```

el resultado de multiplicar el contenido de A (que es el número 5) por 2 se coloca en B que, por tanto, contendrá un 10. Vemos claramente que la principal utilización de la instrucción LET será para memorizar resultados intermedios. Estos resultados podrán ser empleados en sucesivas instrucciones LET o consultados mediante la instrucción PRINT.

RESUMEN

En BASIC existen dos tipos de datos: los numéricos y los textuales también llamados alfanuméricos. Los datos pueden estar en forma de constantes o de variables.

Una constante es un dato cuyo valor permanece fijo y no puede ser modificado.

Las variables son almacenes o memorias para guardar datos. El contenido de una variable puede ser modificado. Las variables tienen un nombre que las identifica y las distingue entre sí. Asimismo, el nombre de la variable indica el tipo de dato que se va a memorizar. No se pueden almacenar datos textuales en variables numéricas ni viceversa.

Para visualizar en la pantalla los datos o los resultados se usa la instrucción PRINT. También se puede emplear para consultar el contenido de las variables. La instrucción PRINT permite realizar operaciones, de tal modo que podemos usar el BASIC como calculadora.

La instrucción LET tiene dos misiones. La primera de ellas es crear el almacén o variable dándole un nombre. La segunda misión es almacenar un dato dentro de la variable. El dato deberá ser de tipo numérico o textual según sea el tipo de la variable. Si la variable que aparece en un LET ya estaba definida, la instrucción va a sustituir el valor antiguo por el nuevo.

EJERCICIOS DE AUTOCOMPROBACION

Complete las siguientes frases:

9. Un programa traductor traduce las instrucciones de un lenguaje de .... a los códigos numéricos que requiere el ordenador.

10. El .... de un lenguaje de programación indica el grado de proximidad que tiene con el lenguaje ordinario.

11. El lenguaje de programación .... es un código simbólico de instrucciones de propósito general para principiantes.

12. Hay dos tipos de memorias, las ROM de .... únicamente y las RAM.

13. La instrucción PRINT se usa para escribir datos en la ....

14. El símbolo «*» indica la operación de ....

15. En BASIC se utiliza el .... para separar los decimales.

16. Las .... son posiciones de memoria donde se almacenan datos.

17. Las variables de tipo textual llevan el símbolo .... al final de su nombre.

18. La instrucción .... permite almacenar datos en las variables.

En las instrucciones siguientes, rodee con un círculo la *V,* si es verdadera, y la *F,* si es falsa.

19. C = LET A	V	F
20. LET A$ = A*3	V	F
21. PRINT C+4	V	F
22. LET A = 4*B − 5	V	F
23. PRINT A$+3	V	F
24. LET E = 2E−6	V	F
25. PRINT A$	V	F
26. LET B = A+1	V	F
27. PRINT A+B$	V	F
28. LET A = PRINT B*2	V	F
29. LET B$ = «Cálculo de Costes»	V	F
30. LET A = LET B = 5	V	F
31. PRINT 1.3567E−2	V	F
32. PRINT «3,1416»	V	F
33. LET 8$ = «HOLA»	V	F

1.6 LOS PROGRAMAS BASIC

1.6.1 Sentencia

Escribamos en el ordenador el siguiente texto:

```
LET A=5 : LET B=3 : PRINT A+B : PRINT A-B
```

y finalicemos como siempre con la tecla de fin de línea.

Aparece en la pantalla del ordenador:

8

2

es decir, la suma y la diferencia de 5 y 3.

Más de una instrucción en una línea

Hemos introducido un elemento nuevo, los dos puntos de separación entre cada una de las instrucciones.

Este signo de puntuación nos permite escribir más de una instrucción con una misma entrada. En el momento que se ha pulsado la tecla de fin de línea, el ordenador ha realizado una instrucción detrás de otra.

Al conjunto de instrucciones que se entran de una sola vez separadas por puntos y acabadas siempre con una pulsación de fin de línea se le denomina *sentencia.*

Instrucción y sentencia

Es importante entender que una sentencia puede contener una instrucción solamente, pero el concepto de instrucción es distinto del de sentencia, ya que una sentencia puede contener varias instrucciones. Por otra parte, una sentencia tampoco es una línea. La línea responde al concepto de anchura básica de la pantalla, y una sentencia puede ocupar más de una línea de pantalla.

La sentencia es el conjunto de caracteres que finalizan con una pulsación de fin de línea.

De hecho, en el ejemplo de sentencia que hemos considerado, se ha introducido una primera idea de programa, pues al pulsar el fin de línea se ha desencadenado la ejecución de varias instrucciones según el orden en que se encuentran, recorriendo la sentencia de izquierda a derecha.



1.6.2 El Programa

Según el apartado 1.1.1. de este capítulo, un programa es una secuencia de instrucciones ejecutadas en un orden preestablecido.

Consideremos un nuevo ejemplo; introduzcamos en el ordenador:

```
10 PRINT "HOLA, BUENOS DIAS"
```

y acabemos con la tecla de fin de línea.

Aparentemente, el ordenador no ha reaccionado. A diferencia de lo que ocurría hasta el momento, el ordenador no ha presentado en ninguna parte el resultado HOLA, BUENOS DIAS.

Número de sentencia

Esta diferencia se basa en el hecho de que ahora hemos escrito un número (el 10) precediendo a la instrucción.

¿Qué debemos hacer ahora para que el ordenador nos imprima la frase?

Deberemos darle la orden de ejecución del programa. Esto se hace escribiendo:

RUN

```
RUN
```

Modo inmediato de ejecución

RUN (del inglés correr, en español se utiliza ejecutar).

Pulsando el fin de línea, vemos que inmediatamente aparece:

```
HOLA, BUENOS DIAS
```

La función del número de línea (en este caso el 10) no es sólo indicar que la instrucción (o instrucciones) que la constituyen forma parte de un programa, sino también el orden en que deben realizarse cada una de las operaciones especificadas.

Si escribimos ahora una nueva línea, con un número inferior:

```
5 PRINT "SOY EL ORDENADOR"
```

y a continuación pulsaremos el fin de línea y a continuación el RUN y de nuevo el fin de línea aparece en pantalla.

```
SOY EL ORDENADOR
HOLA, BUENOS DIAS
```

Vemos por tanto que se ejecuta en primer lugar la sentencia con número de orden más bajo (la sentencia 5 antes que la 10), independientemente del orden en que se han escrito.

Por otra parte, si queremos ver el programa que tenemos en memoria, escribiremos:

LIST

```
LIST .(listar)
```

El efecto de esta orden será hacer aparecer en pantalla:

```
 5 PRINT "SOY EL ORDENADOR"
10 PRINT "HOLA, BUENOS DIAS"
```

Aunque hayamos escrito la línea 5 después de la 10, el ordenador las guarda y ejecuta teniendo en cuenta su número de línea.

Estructura de un programa

El programa esta constituido, pues, por todas las sentencias numeradas que hayamos escrito, que son almacenadas en orden creciente, y podremos verlos en pantalla escribiendo LIST y ejecutarlas escribiendo RUN.

Para numerar las líneas de un programa utilizaremos números enteros, igual o mayores que 1 e inferiores a un valor determinado, y distinto según las máquinas, que puede variar entre 10000 y 65000.

Supongamos ahora que deseamos escribir un nuevo programa. Debemos borrar el que tenemos en memoria. Para ello escribiremos:

```
NEW
```

Esta palabra puede traducirse por «NUEVO». La orden tiene dos efectos, uno observable directamente, y es el que produce el borrado de la pantalla, y otro que no vemos, y es el borrado del programa que tenemos en memoria. Este efecto lo podemos comprobar escribiendo de nuevo la orden:

```
LIST
```

que significa «LISTAR», hacer la lista de las instrucciones del programa.

Comandos para manipular los programas

Observaremos que no aparece nada en pantalla: la memoria está limpia y preparada para introducir un nuevo programa.

Escribiremos entonces:

```
10 LET A = 12
20 LET B = 124
30 LET C = A*B
40 PRINT A + B
50 PRINT C
```

Apretamos después la tecla RUN y la tecla de fin de línea y el ordenador ejecuta el programa. En pantalla aparece 136 (12+124) y 1488 (12×124).

Este programa puede escribirse también de otra forma, para ello escribiremos NEW y:

```
10 LET A = 12  :  LET B = 124  :  LET C = A*B
20 PRINT A + B  :  PRINT C
```

Obsérvese que las instrucciones que se han escrito son exactamente las mismas que antes, sólo que en este caso las hemos escrito agrupadas.

Escriba la orden de RUN y el ordenador ejecutará este programa. Observará que el resultado es el mismo que en el programa anterior.

Por lo tanto, el efecto de ambos programas será el mismo ya que, como hemos dicho, el orden en que están escritas las sentencias en una línea es el orden en que las ejecuta la máquina.

Los valores de número de línea elegidos pueden ser cualesquiera, dentro de los márgenes especificados. De igual manera, puede elegirse cualquier valor como salto entre líneas, aunque no se aconseja utilizar números correlativos por si hay que intercalar líneas.

1.6.3 Adición de una sentencia

Supongamos que además de imprimir la suma de A+B, nos interesa imprimir su diferencia, A−B.

Intercalar números de sentencia

Para ello, basta con escribir:

```
15 PRINT  A-B
```

con lo que al hacer LIST, vemos que el programa ha quedado:

```
10 LET  A = 12  : LET B = 124  : LET  C = A*B
15 PRINT A-B
20 PRINT A+B  :  PRINT C
```

Si ejecutamos el programa mediante la tecla RUN, seguida de la tecla de fin de línea, verá que le da los resultados siguientes: −112 (12−124), 136 (12+124) y 1488 (12×124).

1.6.4 Modificación de una sentencia

Veremos que hay varias formas de hacerlo. De momento, lo haremos de la forma más simple, que consiste en escribir la nueva sentencia, con el mismo número de la que se desea modificar. La sentencia anterior queda automáticamente sustituida por la última escrita.

Así, en el programa anterior, si escribimos:

```
15 PRINT A/B
```

el programa quedará:

```
10 LET A = 12 : LET B = 124 : LET C = A*B
15 PRINT A/B
20 PRINT A+B : PRINT C
```

Borrado de sentencias

De igual manera, si queremos borrar una sentencia, escribiremos el número de la misma, sin nada a continuación. Por ejemplo, para borrar la línea 15, escribiremos 15 y pulsaremos la tecla de fin de línea, con lo que el programa quedará:

```
10 LET A=12 : LET B = 124 : LET C = A*B
20 PRINT A+B : PRINT C
```

Es decir, hemos borrado la sentencia:

```
15 PRINT A/B
```

Hemos visto, pues, que una instrucción se puede escribir numerada, con lo que forma parte de un programa y se ejecuta dentro del mismo, o sin numerar, con lo que su ejecución es inmediata. A estas formas de proceso se les denomina respectivamente ejecución en modo programa y ejecución en modo inmediato.

Por otra parte, hemos visto tres palabras u órdenes (NEW, LIST, RUN) que sirven para manipular (borrar, listar y ejecutar) programas y se utilizan siempre en modo inmediato. Estas órdenes reciben el nombre de *Comandos*.

1.7 INSTRUCCION INPUT

Para asignar un valor a una variable, conocemos la instrucción LET. Así, si en un programa queremos asignar a la variable A el valor 3 e imprimirla, borraremos el programa anterior escribiendo NEW. A continuación podemos escribir, por ejemplo:

```
10 LET A = 3
20 PRINT A*2
```

Escribiendo RUN, veremos que aparece en pantalla el 6.

Si, por la razón que sea, queremos que A tome un valor distinto (5, por ejemplo), deberemos escribir de nuevo:

```
10 LET A = 5
```

y esto deberemos hacerlo cada vez que A deba variar su valor.

Entrar en el ordenador

Existe una forma distinta, y más práctica en según qué ocasiones, de asignar un valor a una variable, que se basa en que la máquina pregunte al usuario (a Ud.) el valor que aquélla debe tomar en cada caso.

Esto puede hacerse mediante la instrucción INPUT (entrada).
Así, podremos escribir:

```
10 INPUT A
```

El listado del programa será entonces:

```
10 INPUT A
20 PRINT A*2
```

Para ejecutar el programa, escribiremos RUN y la tecla de fin de línea, y veremos que la máquina queda a la espera de respuesta por nuestra parte. En algunos casos, aparece un interrogante en la pantalla. Este interrogante indica que la máquina espera que escribamos el valor que debe tomar la variable A, aunque no todas las máquinas utilizan el interrogante, por lo que en cada caso deberá tenerse en cuenta el tipo de máquina.

Así, si queremos darle el valor 9, escribiremos únicamente este número seguido de tecla de fin de línea, y la máquina lo asignará a la variable en cuestión, prosiguiendo el proceso. En este caso, ejecutará la siguiente sentencia, por lo que aparecerá escrito el valor de la variable A multiplicado por 2; por tanto, en pantalla veremos un 18.

Veamos un ejemplo práctico: deseamos escribir un programa para calcular los nuevos precios de rebajas en unos grandes almacenes. Las rebajas serán para todos los productos del 17.5.

Para ello, debemos multiplicar el precio actual por el descuento, y restar el resultado del precio actual. Por ejemplo, para determinar el precio rebajado de un ordenador que vale 64534, con un descuento del 17.5 %, deberemos efectuar la operación:

$$64534 - 64534 * 17.5/100$$

y esto para cada producto distinto.

El programa que nos permitirá hacerlo automáticamente será:

```
NEW
10 INPUT P
20 LET D = 17.5
30 LET NP = P - P*D/100
40 PRINT NP
```

Diferencia entre LET e INPUT

— En la línea 10, se introduce el precio antiguo, que se asignará a la variable P.

— En la línea 20, se asigna el valor del descuento a la variable D.

— En la línea 30, la máquina calcula el nuevo precio (NP), que será igual al precio antiguo (P), menos el 17.5 % del mismo.

— Finalmente, en la línea 40 se da la orden de escribir el nuevo precio calculado, NP.

Cada vez que lo ejecutemos, bastará con indicar el valor del precio anterior, con lo que obtendremos el nuevo precio rebajado de cada producto.

Evidentemente el valor del descuento podría introducirse también con una instrucción INPUT, pero dado que es el mismo para todos los cálculos, es mejor realizar la asignación con una instrucción LET, pues de lo contrario nos veríamos obligados a introducirlo cada vez que ejecutásemos el programa.

Calculemos ahora el nuevo precio del ordenador que costaba 64534. Escribiremos RUN, y la máquina quedará a la espera de que escribamos el precio. Una vez escrito el 64534 pulsaremos la tecla de fin de línea, e inmediatamente aparecerá en pantalla el nuevo precio calculado. En este caso, veremos:

53240.55

Este número puede diferir en decimales de una a otra máquina. Esta diferencia se debe, tal como ya hemos visto, al número de cifras que utiliza cada máquina. Compruebe la utilidad del programa ejecutándolo varias veces, introduciendo en cada caso distintos precios.

Veamos otro ejemplo de programa para calcular el área de diversos triángulos, de los cuales se conocen las dimensiones de la base y de la altura. (Recuerde que el área del triángulo es igual a la base por la altura del mismo, dividido por dos).

```
NEW
10 INPUT BA
20 INPUT AL
30 LET S = BA*AL/2
40 PRINT "EL AREA DEL TRIANGULO ES: "
50 PRINT S
```

En cada ejecución indicaremos en primer lugar el valor de la base, que se asignará a la variable BA, y a continuación el de la altura (se asignará a la variable AL).

Para ejecutar el programa pulsamos la tecla de RUN y la de fin de línea. El ordenador se queda esperando a que le demos el valor de la base. Por ejemplo 3. Después de pulsar la tecla fin de línea, el ordenador seguirá esperando que le demos el valor de la altura. Por ejemplo 4. Pulsando la tecla de fin de línea, el ordenador dará el resultado 6, pues ahora ya tiene todos los datos.

Mensaje que acompaña a la entrada

Para facilitar la utilización a un usuario que no conozca el programa. es de interés indicar lo que se pregunta en cada caso. Para solucionar esto, es posible englobar el mensaje explicativo en la propia instrucción INPUT, escribiéndolo entre comillas.

Así, podemos escribir:

```
NEW
10 INPUT "Entre el valor de la base: ",BA
20 INPUT "Entre el valor de la altura: ",AL
30 LET ...
```

Al escribir RUN, veremos que el mensaje explicativo aparece precediendo al cursor. En pantalla veremos:

Entre el valor de la base:

Contestemos un número cualquiera, por ejemplo 3, y la tecla de fin de línea. Aparecerá ahora en pantalla:

Entre el valor de la altura:

Deberemos contestar otro número, 4 por ejemplo, y después de pulsar el fin de línea, aparecerá en pantalla:

EL AREA DEL TRIANGULO ES:
6

En estos ejemplos hemos utilizado la instrucción INPUT para asignar valores a variables numéricas. Puede utilizarse también de la misma manera para variables alfanuméricas. Veamos un ejemplo: queremos que nuestro ordenador haga un saludo personal a todos los que vienen a verlo. Para ello, borraremos el programa anterior, y escribiremos:

```
10 INPUT "Por favor me quiere escribir su
         nombre",N$
20 PRINT "BUENAS TARDES"
30 PRINT N$
40 PRINT "COMO ESTA USTED?"
```

Al escribir RUN, veremos que aparece en pantalla:

Por favor me quiere escribir su nombre

Contestando un nombre cualquiera, por ejemplo JUAN GARCIA, veremos en pantalla:

```
BUENAS TARDES
JUAN GARCIA
COMO ESTA USTED?
```

La entrada y el tipo de variable

El texto que escribimos tras el interrogante, sean su longitud y contenido los que sean, se asigna a la variable N$.

Al contestar a una instrucción INPUT, debemos ajustarnos al tipo de variable que se nos solicita. Si la variable es de tipo numérico y contestamos

un texto, la máquina no lo aceptará, dando un mensaje determinado, y detendrá el proceso, o bien permitirá que volvamos a escribir el valor correcto dependiendo de qué máquina sea.

El problema no se plantea en el caso inverso, es decir, si contestamos un número a una variable textual, dado que en estas variables el contenido puede ser cualquiera: dígitos, letras, u otro caracter cualquiera.

RESUMEN

Se pueden escribir varias instrucciones en una sola línea separándola por dos puntos. Este conjunto de instrucciones se denomina sentencia. Un programa es un conjunto de varias instrucciones o sentencias.

Para formar un programa se escriben las instrucciones o sentencias precedidas de un número. Este número constituye el número de sentencia e indica la posición donde quedará dicha sentencia en un programa. El BASIC mantiene siempre el orden creciente de números de sentencia aunque éstas se escriban en desorden.

Para ejecutar un programa se emplea el comando RUN. Para obtener un listado de las instrucciones por la pantalla se teclea LIST y para borrar un programa se escribe NEW.

La instrucción INPUT se emplea para introducir datos en una variable en el momento de la ejecución. Al encontrar esta instrucción, el BASIC detiene la ejecución hasta que se teclea una respuesta. El dato que se teclea debe tener el mismo tipo que la variable especificada en el INPUT.

EJERCICIOS DE AUTOCOMPROBACION

Complete las siguientes frases:

34. Una .... del lenguaje de programación BASIC es la acción elemental de ejecución.

35. Varias instrucciones separadas por dos puntos constituyen una ....

36. El comando RUN se utiliza para .... un programa.

37. Para visualizar un programa en la pantalla se utiliza el comando ....

38. La ejecución de un programa sigue el orden establecido por el .... de sentencia.

39. Cuando se repite una sentencia con el mismo número que otra que se ha entrado antes, se .... la que se había entrado primero.

40. Para obtener un listado del programa en la pantalla, se escribe la orden ....

41. Escribiendo NEW se .... la memoria.

42. Para ejecutar un programa se escribe la orden ....

43. La instrucción .... se usa para entrar datos en una variable en el momento de la ejecución del programa.

En las afirmaciones siguientes rodee con un círculo la *V*, si la afirmación es verdadera, y la *F*, si es falsa.

44. La sentencia es un conjunto de instrucciones que caben en una línea de programa. V F

45. Los números de sentencia pueden ser positivos, negativos o incluso cero. V F

46. El orden de ejecución de las sentencias es el mismo que el orden en que se han escrito. V F

47. El comando NEW borra la memoria del ordenador. V F

48. El comando RUN efectúa un listado del programa en pantalla. V F

49. La instrucción INPUT permite escribir un mensaje para aclarar el significado de la variable que se va a entrar. V F

50. Numerar una sentencia es prepararla para que se introduzca en un programa. V F

51. La instrucción INPUT sólo permite entrar variables numéricas. V F

52. Es conveniente numerar las sentencias de uno en uno para ocupar menos espacio. V F

53. Para modificar una sentencia hay que utilizar el comando RUN. V F

54. Si se repite un número de sentencia el lenguaje BASIC rechaza esta sentencia. V F

55. El comando LIST sirve para visualizar el programa en pantalla. V F

56. Para asignar un valor a una variable podemos utilizar la instrucción INPUT o la instrucción LET. V F

57. El comando RUN ejecuta un programa siguiendo el orden de numeración de las sentencias. V F

58. Una instrucción es una acción elemental de ejecución. V F

59. El programa que se da a continuación calcula la longitud de una circunferencia a partir del valor del radio de la misma con la fórmula

$$2 \times 3.1416 \times R$$

3.14.16 es la constante denominada PI y R es el radio.

```
10 INPUT «Entre el radio de la circunferencia», R
20 LET L = 2 * 3.1416 * R
30 PRINT «La longitud de la circunferencia es»
40 PRINT L
```

De las operaciones que se dan a continuación, cuál es la correcta para calcular la longitud de una circunferencia de radio 3.

a) Apretar la tecla RUN. Apretar la tecla fin de línea. Apretar la tecla 3. Apretar la tecla de fin de línea.

b) Apretar la tecla 3. Apretar la tecla de fin de línea. Apretar la tecla RUN. Apretar la tecla de fin de línea.

c) Apretar la tecla RUN. Apretar la tecla 3. Apretar la tecla de fin de línea.

d) Apretar la tecla NEW. Apretar la tecla RUN. Apretar la tecla de fin de línea. Apretar la tecla 3. Apretar la tecla de fin de línea.

60. ¿Qué instrucciones debe añadir al programa anterior para que además calcule e imprima el área de la circunferencia con la fórmula $3.1416 \times R \times R$?
Utilice el ordenador para comprobar los resultados.

Capítulo 2
• Instrucciones REM / PRINT / TAB / INPUT.

ESQUEMA DE CONTENIDO

Objetivos generales	
La instrucción REM [ark] o comentario	Aclaraciones sobre el REM
La instrucción PRINT completa	Impresión de una línea en blanco Cómo evitar el salto del cursor Instrucción PRINT con varias variables El encolumnador TAB (Tabulador)
Más sobre la instrucción INPUT	Entrada de varias variables La cuestión de los separadores Instrucciones para dar la respuesta
Volver a empezar. Instrucción GOTO	Lazos o bucles Contador de vueltas La instrucción de continuación Impresión del número de vueltas Un ejemplo de utilización de un bucle
La sentencia STOP	

2.0 OBJETIVOS GENERALES

En esta lección se verán con más detalle dos instrucciones que ya conoce: la instrucción PRINT y la instrucción INPUT. De esta forma verá ampliadas sus posibilidades de utilización, al mismo tiempo que repasará lo que ya ha aprendido sobre ambas instrucciones.

Por otra parte, aprenderá varias instrucciones nuevas. En primer lugar verá una instrucción que le permite añadir comentarios al listado de los programas.

Estudiará también la primera instrucción de control, que nos permitirá realizar saltos o cambios de secuencia dentro de un programa, y consecuentemente poner en marcha procesos sin fin.

Finalmente, aprenderá a detener un proceso y volver a ponerlo en marcha en cualquier momento y en cualquier punto del mismo.



2.1. LA INSTRUCCION REM[ARK] O COMENTARIO

Vamos a escribir un programa para que el ordenador se despida de una manera personalizada. Escribiremos para ello:

```
NEW
10 INPUT "SU NOMBRE, POR FAVOR: ",N$
20 PRINT "ADIOS"
30 PRINT N$
40 PRINT "HASTA LA VISTA"
```

Si ahora escribimos RUN para ejecutar el programa, veremos que en pantalla aparece la pregunta:

```
SU NOMBRE, POR FAVOR:
```

Contestando ahora un nombre cualquiera, por ejemplo el de nuestro amigo JUAN MARTINEZ, veremos en pantalla:

```
ADIOS
JUAN MARTINEZ
HASTA LA VISTA
```

Añada ahora la siguiente sentencia:

```
5 REM ESTE PROGRAMA SIRVE PARA DESPEDIRSE
```

y a continuación ejecute de nuevo el programa.

¿Observa algún cambio?
Aparece en pantalla:

```
SU NOMBRE, POR FAVOR:
```

Si de nuevo contestamos JUAN MARTINEZ, veremos:

```
ADIOS
JUAN MARTINEZ
HASTA LA VISTA
```

El comentario. Sentencias que no se ejecutan

Vemos pues que en cuanto a la ejecución no ha variado absolutamente nada. Sin embargo, nosotros hemos añadido una sentencia, la número 5, cosa que efectivamente podemos comprobar viendo el listado completo del programa. Para ello, debemos escribir:

```
LIST
```

Al pulsar la tecla de fin de línea, vemos que aparece en pantalla:

```
 5 REM ESTE PROGRAMA SIRVE PARA DESPEDIRSE
10 INPUT "SU NOMBRE, POR FAVOR: ",N$
20 PRINT "ADIOS"
30 PRINT N$
40 PRINT "HASTA LA VISTA"
```

¿De qué sirve entonces la sentencia que hemos añadido?

Vemos que contiene una palabra especial: REM, y a continuación un mensaje que nos da una indicación sobre el programa.

La finalidad de esta instrucción es intercalar comentarios en el programa, de forma que al leer el listado nos faciliten la comprensión del mismo.

Para ello basta escribir en la posición que convenga —que viene dada, como ya sabemos por el número de sentencia—, la palabra REM, y a continuación el mensaje que se quiera.

El número de comentarios no altera la ejecución

Podemos añadir tantas instrucciones REM como creamos conveniente, ya que, como hemos visto, no afectan en absoluto a la ejecución del programa.

```
7 REM EN PRIMER LUGAR PREGUNTA UN NOMBRE
8 REM Y A CONTINUACION IMPRIME CUATRO LINEAS
```

al escribir LIST veremos que el programa queda:

```
 5 REM ESTE PROGRAMA SIRVE PARA DESPEDIRSE
 7 REM EN PRIMER LUGAR PREGUNTA UN NOMBRE
 8 REM Y A CONTINUACION IMPRIME CUATRO LINEAS
10 INPUT "SU NOMBRE, POR FAVOR: ",N$
20 PRINT "ADIOS"
30 PRINT N$
40 PRINT "HASTA LA VISTA"
```

Escribiendo RUN una vez más, veremos que sigue sin producirse ninguna variación con respecto al mismo programa sin las instrucciones REM.

Las instrucciones REM sirven únicamente al programador para facilitarle la comprensión del listado y, en general, para documentar el programa. Así, si Ud. tiene un archivo de programas puede poner el título del programa mediante la instrucción REM, la fecha en que lo realizó, etc. De esta manera, se le facilitará cualquier consulta posterior.



2.1.1. Aclaraciones sobre el REM

No debe confundirse una instrucción REM (no tiene ningún efecto en la ejecución del programa) con la instrucción PRINT (recordemos que ésta es una orden de escritura). Para ver claramente la diferencia entre ambas, escribamos (sin número de sentencia):

```
REM Esto es una prueba
```

Al pulsar la tecla de fin de línea no se observa respuesta alguna.

Escriba ahora, también sin numerar:

```
PRINT "Esto es una prueba"
```

y pulse la tecla de fin de línea.

¿Qué observa en esta ocasión? La máquina realiza una acción; aparece en pantalla:

```
Esto es una prueba
```

La instrucción PRINT da una orden de escritura, que la máquina debe ejecutar.

Observe que el texto que acompaña a esta instrucción debe escribirse según unas normas determinadas (en este caso lo hemos escrito entre comillas). Estas normas ya se han explicado anteriormente, y obedecen al hecho de que la máquina debe «entender» qué es lo que ha de escribir, debe interpretar la orden, antes de obedecerla. Si queremos escribir el valor de una variable, escribiremos su nombre exacto, y de forma correcta. Si queremos que nos imprima un texto fijo deberemos escribirlo entre comillas, etc.

Por ejemplo: PRINT A, PRINT V$, PRINT «HOLA», etc.

El BASIC prescinde del texto que sigue a un REM

Por el contrario, la *máquina prescinde de lo que está escrito a continuación de la palabra REM.* Por esta razón, el texto que se escriba es totalmente libre. Cualquier línea de programa que empiece por la palabra REM será ignorada.

Para comprobarlo, escribamos de nuevo las líneas de programa 10 y 30, pero con la palabra REM delante:

```
10 REM INPUT "SU NOMBRE, POR FAVOR: ",N$
20 PRINT "ADIOS"
30 REM PRINT N$
40 PRINT "HASTA LA VISTA"
```

Si ahora ejecutamos el programa, observaremos que la máquina no espera que contestemos el nombre; escribe directamente:

```
ADIOS
HASTA LA VISTA
```

No se ha realizado la pregunta, y ha escrito sólo dos líneas, como si tuviera únicamente dos instrucciones PRINT. Esto se debe a la presencia de la palabra REM al principio de las líneas correspondientes. Estas siguen estando —puede comprobarlo escribiendo LIST—, pero no son ejecutadas.

Escriba ahora el programa de nuevo, pero agrupando las instrucciones:

```
10 INPUT "SU NOMBRE, POR FAVOR: ",N$
20 PRINT "ADIOS" : PRINT N$ : PRINT "HASTA
   LA VISTA"
```

Compruebe el funcionamiento, escribiendo RUN. El programa preguntará el nombre, y una vez contestado, aparecerá en pantalla:

```
ADIOS
JUAN MARTINEZ
HASTA LA VISTA
```

El efecto es el mismo que cuando habíamos escrito una sola instrucción por línea. Pero escriba ahora de nuevo:

```
20 PRINT "ADIOS" : REM PRINT N$ : PRINT
   "HASTA LA VISTA"
```

Ejecutémoslo ahora. Observamos que se realiza la pregunta, pero una vez contestada, aparece únicamente en pantalla:

```
ADIOS
```

¿Por qué no se ejecutan las dos instrucciones restantes?

Hemos intercalado la palabra REM al inicio de la segunda. Tal como hemos dicho, el efecto de esta palabra al principio de una instrucción es anular —a efectos de ejecución— todo lo que viene a continuación. Por tanto, la máquina ignora las dos instrucciones que siguen, por esta razón no escribe el nombre ni el texto «HASTA LA VISTA».

Atención:

La posición de la palabra REM es al principio de la instrucción, esté ésta situada donde esté. Si es la primera instrucción de una sentencia, escribiremos REM en primer lugar, a continuación del número de sentencia. Si no es la primera instrucción de la sentencia, deberá escribirse inmediatamente después de los dos puntos de separación entre instrucciones.

El comentario al principio de sentencia anula todas las instrucciones de la sentencia

Recordemos el ejemplo visto en el epígrafe 1.7. del capítulo anterior. En él se calculan los nuevos precios de rebajas de unos grandes almacenes. Recordemos que se introduce el precio de cada producto, se asigna el % de descuento, se calcula el nuevo precio y se imprime el resultado. El listado del programa es el siguiente:

```
10 INPUT P
20 LET D = 17.5
30 LET NP = P - P*D/100
40 PRINT NP
```

A continuación del listado, habíamos escrito la explicación de cada sentencia. Utilizando la instrucción REM, podremos incluir estas explicaciones en el propio listado, de la siguiente manera:

```
 5 REM Calculo del precio rebajado de un
   articulo
10 INPUT P : REM Introduccion del precio
   antiguo
20 LET D = 17.5 : REM Asignacion del valor
   del descuento
30 LET NP = P - P*D/100 : REM Calculo del
   precio rebajado
40 PRINT NP : REM Impresion del resultado
```

A efectos de ejecución el programa no ha variado en absoluto, pero a

efectos de comprensión y de lectura del mismo, nos resulta mucho más sencillo.

En resumen, vemos pues que la instrucción REM nos permite incluir cualquier texto en el programa para documentarlo, aclarar algún punto, etc. La necesidad de hacerlo quizá no quede todavía muy clara, ya que hasta el momento hemos visto pocos programas, y muy cortos. Más adelante haremos programas más largos, y podremos almacenarlos —ya veremos cómo—. Se verá entonces que tener información de lo que hace cada programa y del funcionamiento de algún punto de mayor complejidad no sólo es útil, sino necesario y en ocasiones, incluso, imprescindible.

2.2 LA INSTRUCCION PRINT COMPLETA

Conocemos ya la instrucción PRINT, la hemos utilizado muchas veces, y sabemos para qué sirve.

Sin embargo, esta instrucción ofrece mayores posibilidades que las conocidas. Vamos a ver algunas de ellas.

2.2.1 Impresión de una línea en blanco

Si observamos con atención el efecto de una instrucción PRINT, veremos que es doble: por una parte, realiza una escritura y por otra, produce el salto del cursor a la línea siguiente.

Así, escribamos:

```
PRINT "HOLA"
```

pulsemos la tecla de final de línea y observemos el resultado. Veremos que en pantalla aparece la palabra

```
HOLA
```

y que el cursor salta a la siguiente línea, justo debajo de la primera letra.

Esto puede aprovecharse para escribir líneas en blanco. Para ello, bastará escribir:

```
PRINT
```

Al dar la orden de escritura sin indicar nada que escribir, la orden es obedecida al pie de la letra: se escribe lo que indica la instrucción, es decir, nada, y el cursor salta a la línea siguiente. El efecto global es pues la impresión de una línea en blanco.

Para saltar líneas se utiliza la instrucción PRINT sola

Si se desean varias líneas en blanco, se escribirán varias instrucciones PRINT. Comprobémoslo escribiendo:

```
NEW
10 PRINT "INICIO"
20 PRINT
30 PRINT
40 PRINT
50 PRINT "FINAL"
```

La ejecución de este programa hará que aparezca en pantalla:

Línea 1:	INICIO
Línea 2:	
Línea 3:	
Línea 4:	
Línea 5:	FINAL

Según puede observarse, entre las palabras INICIO y FINAL, se han escrito tres líneas en blanco.

Repasando lo que ya se ha visto anteriormente, en la misma línea se pueden escribir varias instrucciones, separadas por el símbolo dos puntos (:).

El programa anterior podrá entonces escribirse:

```
NEW
10 PRINT "INICIO"
20 PRINT : PRINT : PRINT
30 PRINT "FINAL"
```

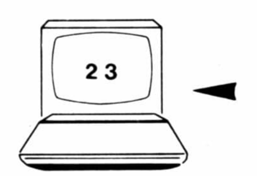
Al ejecutarlo observaremos exactamente los mismos efectos que en el caso anterior.

Y aun podría haberse escrito:

```
NEW
10 PRINT "INICIO":PRINT:PRINT:PRINT:PRINT
   "FINAL"
```

observaremos que al ejecutarlo produce los mismos resultados que en las dos formas anteriores.

De esta última forma hemos obtenido un programa más compacto. Sin embargo, no debe olvidarse que el programa ha de ser fácil de leer; siempre ha de tenerse presente la necesidad de claridad, aunque el listado sea más largo, o quede menos compacto. El efecto de cara a la ejecución por la



máquina no varía en absoluto, y en cambio la manipulación del programa (lectura, corrección de errores, modificaciones, etc.) se hace más fácil para el programador.

2.2.2 Cómo evitar el salto del cursor

Como ya sabemos, cada instrucción PRINT imprime una línea. Así, en un programa con tres instrucciones PRINT se imprimirán tres líneas. Para comprobarlo nuevamente, escribiremos:

```
NEW
10 INPUT N$
20 PRINT "HOLA..."
30 PRINT N$
40 PRINT "QUE HAY ?"
```

Ejecutémoslo ahora y contestemos un nombre cualquiera a la pregunta, por ejemplo FERNANDO. Veremos en pantalla:

```
HOLA...
FERNANDO
QUE HAY ?
```

Pero, ¿qué pasa si queremos escribir el resultado como una sola frase, es decir en una sola línea? ¿Cómo podemos hacerlo?

La instrucción PRINT sin saltar de línea

Tal como hemos visto, cada instrucción PRINT imprime una nueva línea: escribe el texto que corresponde y salta a la línea siguiente. Sin embargo, hay una forma de impedir que después de escribir el texto, el cursor pase a la siguiente línea. Para ello, basta con escribir un punto y coma (;) al final de la línea.

Vamos a comprobarlo. Escribamos:

```
20 PRINT "HOLA...";
```

y a continuación ejecutemos el programa escribiendo RUN. Después de contestar el nombre, veremos en pantalla:

```
HOLA...FERNANDO
QUE HAY ?
```

El cursor, después de escribir el primer texto, no ha avanzado a la línea siguiente, quedando justo a continuación del mismo. El nombre, pues, aparece a partir del final del primer texto.

El punto y coma no desplaza al cursor después de una escritura

Ahora, para conseguir que después del nombre no salte a la línea siguiente, deberemos añadir un punto y coma (;) al final de la línea PRINT que lo escribe.

Escribiremos pues:

```
30 PRINT N$;
```

Ejecutando ahora el programa, después de contestar el nombre, veremos en pantalla:

```
HOLA...FERNANDOQUE HAY ?
```

El cursor no ha saltado, escribiendo el último texto justo a continuación del anterior, según indica el punto y coma. Si deseamos que entre ellos aparezca un espacio en blanco, deberemos incluirlo en el texto. Así escribiremos, dejando un espacio en blanco después de las primeras comillas:

```
40 PRINT " QUE HAY ?"
```

Lo mismo podemos hacer si queremos que entre los puntos suspensivos y el nombre quede un espacio. En este caso escribiremos también, un espacio en blanco, después de los puntos suspensivos.

```
20 PRINT "HOLA... ";
```

Si ejecutamos ahora el programa, el resultado será:

```
HOLA... FERNANDO QUE HAY ?
```

> El símbolo punto y coma (;) al final de una instrucción PRINT, evita el salto del cursor a la línea siguiente, e impide el avance del mismo.

Hay además otra forma de hacer que el cursor no pase a la línea siguiente, o lo que es lo mismo, escribir una sola línea aun teniendo varias instrucciones PRINT.

La coma tiene un comportamiento intermedio entre el punto y coma y el salto de línea

El otro símbolo que nos permite hacer esto es la coma (,). Situándola al final de una instrucción PRINT, veremos que evita el salto del cursor a la línea siguiente, aunque su efecto global no es el mismo que el punto y coma.

Vamos a ver cómo actúa. Modificamos para ello el programa, escribiendo de nuevo:

```
30 PRINT N$,
```

con lo que hemos sustituido el punto y coma (;) por una coma (,).

Ejecutando ahora el programa, después de escribir el nombre, veremos en pantalla:

```
HOLA... FERNANDO            QUE HAY ?
```

Observamos en este caso, que entre el nombre y la palabra QUE, ha quedado un cierto número de espacios en blanco. Volvamos ahora a ejecutar el programa, pero respondamos un nombre distinto, por ejemplo ALI. Observaremos ahora en pantalla:

```
HOLA... ALI   QUE HAY ?
```

El número de espacios entre el nombre y la palabra QUE es ahora distinto.

¿Cuál es la causa de esta diferencia?

El número de espacios en blanco que quedan a continuación de la coma no es un número al azar, ni depende del tipo de nombre que escribamos. Sí depende, sin embargo, de su longitud. Vamos a ver por qué.

La coma actúa como si apretáramos la tecla de Tabulación en una máquina de escribir. Podemos considerar que la pantalla está dividida —naturalmente de forma invisible— en diversas «zonas de tabulación». (Fig. 1) El número de espacios entre cada zona de tabulación puede variar según la máquina que utilicemos. Este número de espacios para nuestra máquina lo conocemos por las «Prácticas con el microprocesador».

Las comas nos sitúan en las distintas zonas de tabulación de la pantalla

Sin embargo, en todas las máquinas, el efecto de la coma es el salto a la siguiente zona de tabulación.

Cuando ejecutamos el programa que tenemos en memoria, y contestamos FERNANDO, suponiendo que las zonas de tabulación están en la columna 14, 28, etc. al escribir HOLA... FERNANDO, la última O queda en la columna 16, la coma hará que el cursor salte a la siguiente zona de tabulación, es decir, a la columna 28. Escribirá en ella el resto del texto. Este empieza por un espacio en blanco, por tanto, la letra Q aparecerá en la columna 29. (Fig. 2.)

Volvamos ahora a ejecutar el programa, pero contestemos el nombre de ALI. La I última se situará en la columna 11, la coma nos hará saltar a la siguiente zona, que se encuentra en la columna 14. En esta se escribirá el espacio en blanco, y la letra Q quedará en la columna 15. (Fig. 2.)

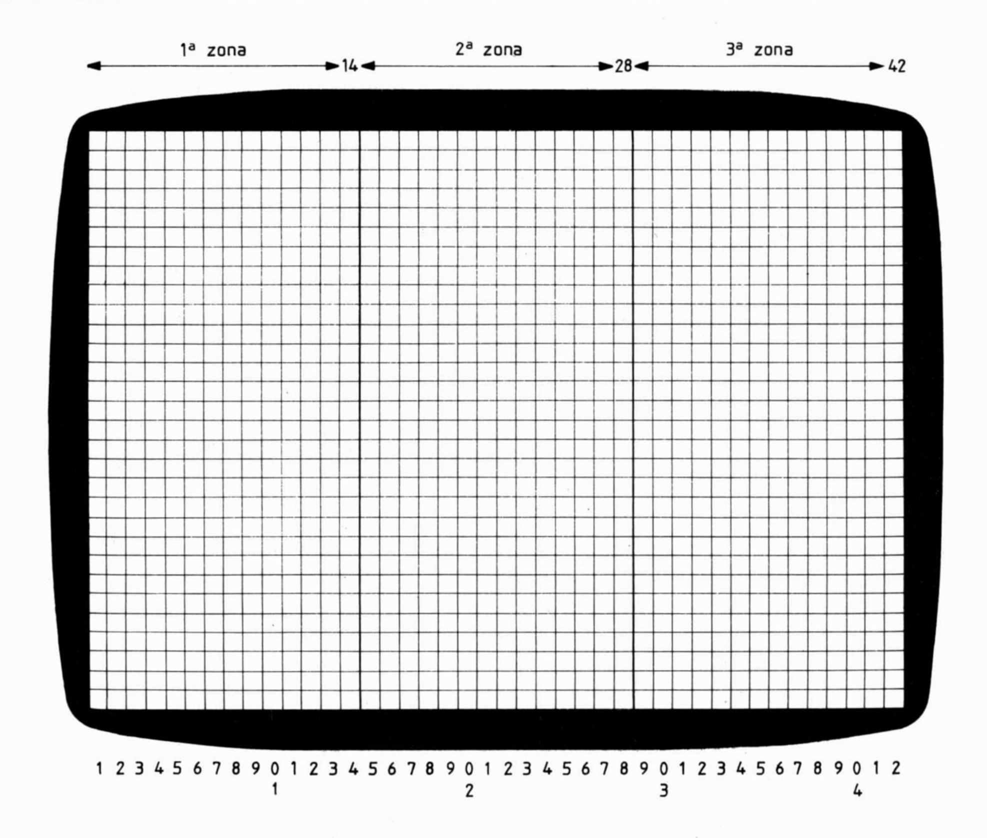


Figura 1. Pantalla con tres zonas de tabulación de 14 columnas cada una

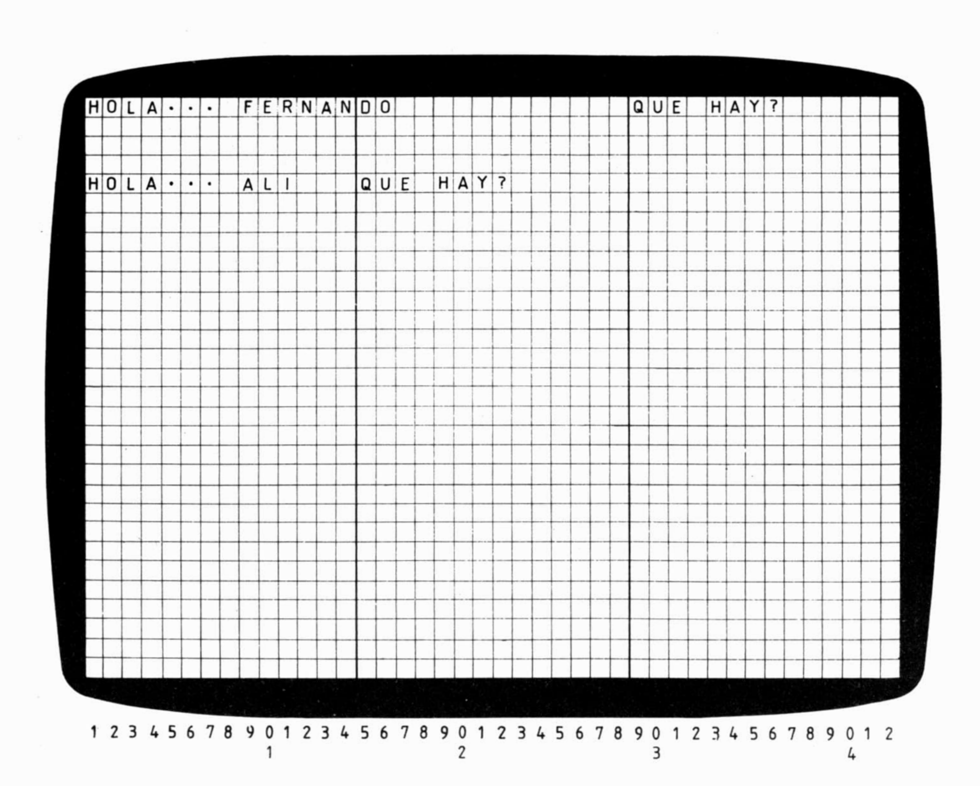


Figura 2. Colocación del texto después de PRINT seguido de coma

Ejecutándolo para diversos nombres, podrá ir viendo el efecto de la coma. Por ejemplo, conteste un nombre de 4 letras (como PEPE) y observará que la palabra QUE se escribe en la misma posición. Sin embargo, si respondemos una palabra de 7 letras (por ejemplo TEODORICO), la última letra queda en la columna 17, por lo que el texto siguiente se empezará a escribir en la siguiente zona de tabulación, es decir la columna 28. (Recuerde que el texto empieza con un blanco, la Q quedará pues en la columna 29.)

Cuando se salta más allá de la última zona de tabulación se salta de línea

Finalmente añadir, que cuando no existe la siguiente zona de tabulación en la línea que estamos escribiendo, el ordenador salta a la primera columna de la línea siguiente. Por ejemplo, al realizar la instrucción siguiente:

```
PRINT "1","2","3","4","5"
```

El 1 aparecerá en la columna 1, el 2 aparecerá en la columna 14, el 3 aparecerá en la columna 28, el 4 aparecerá en la 1 y el 5 aparecerá en la 14. De esta forma podemos escribir líneas sucesivas encolumnadas.

Hemos visto pues que en la coma y el punto y coma nos permiten controlar el salto del cursor. De esta forma, puede obtenerse una sola línea como resultado de varias instrucciones PRINT.

2.2.3 Instrucción PRINT con varias variables

La instrucción PRINT nos permite escribir más de una variable cada vez. Para ello, las variables se escriben separándolas ya sea con un punto y coma o por una coma. El efecto en cada caso es el descrito en el apartado anterior.

Así, podemos sustituir las tres instrucciones PRINT del programa anterior por una única instrucción, que incluya los dos textos y la variable.

Podemos pues escribir:

```
NEW
10 INPUT N$
20 PRINT "HOLA... ";N$;" QUE HAY ?"
```

Observe que ya se han incluido los blancos después de los puntos suspensivos y antes de la Q; por lo tanto, estas instrucciones no son exactamente iguales a las del programa anterior. Si lo ejecutamos y contestamos un nombre cualquiera, por ejemplo PEPE, veremos en pantalla:

```
HOLA... PEPE QUE HAY ?
```

El punto y coma de separación hace que el cursor no se mueva de

sitio. Como ya se ha visto en la sección anterior, evita el avance; por tanto, quedan los textos escritos uno a continuación del otro.

Escribamos ahora una coma en lugar del punto y coma de separación:

```
20 PRINT "HOLA... ",N$," QUE HAY ?"
```

si ejecutamos ahora el programa, después de contestar el nombre, veremos:

```
HOLA...       PEPE          QUE HAY ?
```

En este caso, la presencia de las comas de separación ha hecho que cada texto se escriba en una zona de tabulación. Así, HOLA... queda en la primera posición, la coma hace que PEPE quede en la segunda zona de tabulación, y la coma que se encuentra a continuación produce el salto del cursor a la siguiente zona de tabulación, donde se escribe un espacio en blanco, quedando la letra Q en la columna 29.

Es importante recordar que en el ordenador que utilizamos como ejemplo en este capitulo tiene las zonas de tabulación en las columnas 14 y 28. Sin embargo, en otro ordenador pueden ser diferentes, incluso que la pantalla tenga una sola zona de tabulación, por ejemplo, en la columna 17. En este caso, el ejemplo, ocuparía dos líneas ya que la tercera zona de tabulación sería la primera columna de la línea siguiente.

Esto es un inconveniente, pues un programa que en un ordenador nos da un encolumnado perfecto de una tabla, en otro ordenador, este encolumnado puede destruirse totalmente. A pesar de estos inconvenientes, los fabricantes de ordenadores no se han puesto de acuerdo globalmente en utilizar las mismas zonas de tabulación.

De esta forma pues, podemos utilizar una misma instrucción PRINT para escribir varias variables o constantes, ya sean de tipo numérico o alfanumérico, separándolas por punto y coma o por coma, según interese en cada caso.

En una misma instrucción PRINT se pueden utilizar varias variables

Vamos a efectuar algunas pruebas más. Así, si escribimos (sin numerar):

```
PRINT "NUMERO:";3;"LETRA:";"A";"-FIN-"
```

Cabe esperar del efecto que tiene el punto y coma que el resultado en pantalla sea:

```
NUMERO:3LETRA:A-FIN-
```

ya que sabemos que el punto y coma no mueve en absoluto el cursor, y,

por lo tanto, el 3 aparecerá escrito pegado a los dos puntos del rótulo NUMERO; y a la L del rótulo LETRA.

Sin embargo, en otros BASIC veremos en pantalla:

```
NUMERO: 3 LETRA:A-FIN-
```

Observe que antes y después del número hay un espacio en blanco, que nosotros no hemos escrito, y que no ha quedado antes y después de la letra, habiendo utilizado el mismo símbolo de separación (el punto y coma) en ambos casos.

Esto se debe a que el 3 es una constante numérica.

En este tipo de Basic las constantes y variables numéricas las escribe con un espacio en blanco delante y un espacio en blanco detrás.

Note, sin embargo, que si hubiéramos escrito el número 3 entre comillas, este tipo de BASIC no nos hubiera dejado los espacios, ya que el 3 entre comillas no tiene significado de número, sino de texto.

Los valores numéricos van precedidos y acabados con un caracter en blanco

Comprobémoslo escribiendo (sin numerar):

```
PRINT "NUMERO:";"3";"-FIN-"
```

Veremos ahora en pantalla:

```
NUMERO:3-FIN-
```

Cuando se escriben constantes y variables textuales el efecto del punto y coma es el mismo para todos los dialectos del BASIC.

Estos pequeños detalles hacen que todos los BASIC sean muy similares, pero a la vez el comportamiento de una instrucción puede ser distinta, y, por lo tanto, programas iguales pueden comportarse de manera distinta, ejecutados con lenguajes BASIC distintos.

Escriba ahora la misma instrucción, pero utilizando la coma de separación, es decir:

```
PRINT "NUMERO:","3","-FIN-"
```

Veremos en pantalla:

```
NUMERO:        3              -FIN-
```

quedando la N primera en la primera columna, el 3 en la primera zona de tabulación y el primer guión en la siguiente zona de tabulación. En nuestro

caso la primera zona de tabulación significa la columna 14 y la siguiente significa la 28.

Escribamos ahora:

```
PRINT "NUMERO:",3,"-FIN-"
```

El efecto de esta instrucción depende del tipo de BASIC. En el caso de que trabajemos con un BASIC que a las constantes numéricas añada blancos, uno delante y otro detrás.

```
NUMERO:        3             -FIN-
```

La N primera quedará en la primera columna, pero el número 3 en la segunda columna de la primera zona de tabulación, ya que se ha escrito como constante numérica, no textual, quedando por tanto con un espacio delante y otro detrás. Atención al hecho de que el primer guión queda en la misma posición que en la prueba realizada antes. Esto se debe a que, aunque después del número se escriba un blanco de más, la presencia de la coma a continuación produce el salto a la siguiente zona, independientemente del punto intermedio en que hemos terminado.

El guión quedará pues en la columna en la siguiente zona de tabulación, en nuestro caso la columna 28.

En cambio, si trabajamos con un BASIC que no añada blancos a la escritura de constantes o variables numéricas, el resultado de ambas instrucciones es idéntico.

Resumiendo, podemos decir:

— El efecto del punto y coma es evitar el movimiento del cursor, tanto si se utiliza como separador entre variables, como si se encuentra al final de la instrucción PRINT

— El efecto de la coma es situar el cursor en la siguiente zona de tabulación, dentro de la misma línea, tanto si se usa como separador como si está al final de una instrucción PRINT

Si el texto que se desea imprimir no cabe en una sola línea de pantalla, se escribe la línea completa y el cursor pasa automáticamente a la línea siguiente, manteniéndose los efectos del punto y coma y de la coma según lo habitual.



2.2.4 El encolumnador TAB (Tabulador)

¿Cómo podremos escribir un texto en el centro de la pantalla? Y situarlo en el extremo de la línea? ¿Y en la columna 24?

Ya hemos visto hasta ahora, que podemos escribir el texto precedido de tantos blancos como sean necesarios para que el texto se sitúe en la posición que interesa.

Así, para escribir HOLA empezando en la columna 24, deberemos escribir:

```
PRINT "                        HOLA"
```

para escribirlo en la 63, deberemos dejar 62 blancos, para hacerlo en la 40, dejaremos 39 blancos, etc.

Esto nos obliga a escribirlos y contarlos todos.

El encolumnador TAB permite dirigirnos a la columna que queramos

Este trabajo puede ahorrarse gracias al encolumnador TAB. Este nos permite indicar directamente en qué columna queremos escribir, y sitúa el cursor en la misma. Para ello, se escribe TAB, y a continuación entre paréntesis el número de columna que se quiera. Así, se escribirá TAB(12), TAB(24), TAB(65)...

Esto se utiliza dentro de la instrucción PRINT.

Veamos cómo funciona con un ejemplo, escribamos:

```
PRINT TAB(18);"ESTOY AQUI"
```

en pantalla veremos, en la columna 18, pues la primera columna es la cero.

```
                  ESTOY AQUI
```

El encolumnador puede utilizarse tantas veces como haga falta dentro de una misma instrucción PRINT. Así, se puede escribir:

```
PRINT TAB(5);"COL. 5";TAB(15);
 "COL. 15";TAB(22);"COL. 22"
```

el efecto en pantalla será:

```
     COL. 5    COL. 15  COL. 22
```

El número indica la columna en que debe escribirse contando a partir del principio.

El TAB no nos permite retroceder

Debe tenerse en cuenta que el cursor no puede retroceder. Así, si hemos escrito ya después de una columna, no se puede utilizar un TAB con un número inferior al de aquella.

Así, si escribe:

```
PRINT "ESTO ES UN";TAB(5);"ERROR"
```

está cometiendo efectivamente un error, pues la N de UN se escribirá en la décima columna, no tiene sentido entonces querer situar la palabra ERROR en la columna quinta.

Atención pues:

Los saltos con el encolumnador deben ser siempre hacia adelante, el cursor no puede retroceder.

Cuando por equivocación o no, colocamos una función TAB para ir a una columna que hemos sobrepasado, el efecto consiste en ir a la columna que le pedimos en la línea siguiente.

RESUMEN

La instrucción REM se utiliza para añadir comentarios a un programa, de forma que se facilite su lectura y comprensión.

Por otra parte, la instrucción no tiene ningún efecto sobre la ejecución del programa.

El programador debe utilizar esta instrucción para aclarar los diversos aspectos de lo que hace el programa, para separar partes del programa bien diferenciadas y, en general, toda la documentación necesaria para interpretar el programa de una manera más cercana al lenguaje natural.

Los comentarios son necesarios, pues, cuando se programa se tiene el problema a resolver muy claro. Después, cuando pasa el tiempo y tiene que volverse al programa, estos detalles importantes se han olvidado y son necesarios los comentarios para evitar estos problemas.

La instrucción PRINT se puede utilizar para escribir líneas en blanco, es decir saltar líneas. Para ello se escribe la instrucción PRINT sin ninguna constante ni variable detrás de la palabra PRINT.

La instrucción sirve también para escribir diversas variables y constantes a la vez, es decir, con una única instrucción PRINT.

Para escribir varias cosas a la vez, cada una de las variables o constantes a escribir va separada de la siguiente, mediante dos signos de puntuación: la coma (,) y el punto y coma (;).

Cuando se utiliza como separador el punto y coma, no se produce ninguna separación entre las variables o constantes a escribir. Cuando alguna de ellas es numérica, los dígitos de los números van precedidos por un espacio en blanco y finaliza también con un espacio en blanco.

Sin embargo este comportamiento no lo siguen todos los dialectos del lenguaje BASIC. Es necesario experimentar para averiguar el comportamiento de un BASIC en concreto.

Si el separador es una coma las separaciones de las dos variables depende de la situación del cursor en aquel momento. Las varia-

bles se encolumnan en unas columnas definidas de antemano. Estas columnas se denominan inicios de las zonas de tabulación.

El número de zonas de tabulación y la situación de los inicios de estas zonas de tabulación dependen del dialecto del lenguaje BASIC que se utilice. Es necesario experimentar con el BASIC concreto que se trabaja.

En cualquier caso, si una variable no cabe en una línea, la escritura sigue en la línea siguiente, incluso dividiendo el texto a escribir en dos trozos, el primero que se escribe en la línea superior y el segundo en la línea inferior.

Si se termina una instrucción PRINT con un punto y coma, el efecto se acumula para la próxima instrucción PRINT.

También si una instrucción PRINT se termina con una coma el efecto se acumula para la siguiente instrucción.

En cualquier caso, el hecho de que las zonas de tabulación sean distintos para cada dialecto del BASIC puede ocasionar que un programa funcione distinto en una máquina que en otra.

El encolumnador TAB es un instrumento que permite situarnos en la columna que deseemos de la pantalla. El número de espacios necesarios entre el inicio de la línea y el lugar donde queremos empezar a escribir se encierra entre paréntesis detrás de la palabra TAB. Por ejemplo TAB (23), inicia a escribir en la columna 24.

Cuando estamos en una columna y colocamos una instrucción TAB con un número de espacios menor, el BASIC estándard da un error. En otro tipo de BASIC el siguiente carácter a escribir se sitúa en la línea siguiente.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

1. Cuando se quiere poner una observación del funcionamiento de un programa se utiliza la instrucción ....

2. La instrucción REM deja sin efecto todas las .... que puedan haber desde su colocación hasta el final de la sentencia.

3. Para escribir una línea en blanco hay que poner la instrucción .... sin ninguna variable ni constante.

4. Los .... son los signos de puntuación que sirven para especificar la lista de variables o constantes que debe escribir la instrucción PRINT.

5. Los dos separadores que utiliza el BASIC son la .... y el punto y coma.

6. El efecto de la .... consiste en saltar a la siguiente zona de tabulación, o a la siguiente línea si se está en la última zona de tabulación.

7. Cuando se quieren escribir dos textos pegados uno junto al otro se utiliza como separador el ....

8. Para poder escribir en una columna determinada se utiliza el ....

9. El encolumnador TAB tiene como efecto dirigirnos a la .... siguiente del número que colocamos entre paréntesis.

10. El efecto de los separadores se .... en la instrucción PRINT siguiente cuando están al final de la instrucción.

Encierre en un círculo las respuestas que correspondan a la alternativa correcta.

11. La instrucción REM se utiliza para:

   a) Documentar los programas.
   b) Imprimir una línea de resultados.
   c) Entrar un dato en el programa.
   d) Ejecutar un programa.

12. Cuando la instrucción REM está situada al inicio de una sentencia, las demás instrucciones de la sentencia:

   a) Se ejecutan normalmente.
   b) Se ejecutan sólo si son sentencias PRINT.
   c) No se ejecutan en ningún caso.
   d) Se ejecutan sólo si hay otra instrucción REM.

13. En un ordenador de 40 columnas por línea que tiene las zonas de tabulación cada 15 columnas, en qué columna se escribirá la letra Z de la instrucción

PRINT «A»; «B», «L», «Z»

a) En la primera de la línea siguiente.

b) En la columna 31.

c) En la columna 30.

d) En la columna 16.

14. Los valores numéricos se escriben en el BASIC estándard con un espacio en blanco que les precede y un espacio en blanco que les sigue:

a) Si se escriben en una instrucción PRINT que contenga más de una variable.

b) Cuando van después de un punto y coma.

c) Nunca.

d) Siempre.

15. En un ordenador de 32 columnas por línea con zonas de tabulación de 10 columnas en 10 columnas, dónde se escribe la letra K cuando se ejecuta la sentencia siguiente:

PRINT «A»; «D»; : PRINT «K»

a) En la primera posición de la segunda línea.

b) En la tercera posición de la misma línea.

c) En la tercera posición de la siguiente línea.

d) En la columna 11.

16. En un ordenador de 32 columnas por línea, con zonas de tabulación de 16 columnas, dónde se escribe el número 1 cuando se ejecuta la sentencia siguiente:

PRINT «O», : PRINT «1»

a) En la columna 2 de la misma línea.

b) En la columna 17 de la misma línea.

c) En la columna 1 de la línea siguiente.

d) En la columna 17 de la línea siguiente.

17. ¿Qué instrucción debemos escribir para colocar la palabra HOLA en la columna 13?

a) PRINT «HOLA»

b) PRINT TAB(13); «HOLA»

c) PRINT TAB(12); «HOLA»

d) PRINT TAB(14); «HOLA»

18. Cuando escribimos TAB(23) y estamos en la columna 30 en un ordenador de 30 columnas, en qué columna continuamos escribiendo:

a) En la 24 de la misma línea.

b) En la 23 de la misma línea.

c) En la 24 de la línea siguiente.

d) En la primera de la línea siguiente.

19. En un ordenador de 32 columnas y de zonas de tabulación de 16 columnas en 16 columnas. Cuántos espacios en blanco habrá entre la última letra de HOLA y la primera de ADIOS al ejecutar la instrucción siguiente:

PRINT «HOLA», «ADIOS»

a) 10.

b) 11.

c) 12.

d) 13.

20. Cómo queda la línea al ejecutar la instrucción siguiente:

PRINT «SOY EL»; : PRINT «ORDENADOR»

a) SOY ELORDENADOR

b) SOY EL
ORDENADOR

c) SOY EL ORDENADOR

d) SOYEL
ORDENADOR



2.3 MAS SOBRE LA INSTRUCCION INPUT

Ya hemos visto anteriormente cómo se escribe y para qué sirve la instrucción INPUT. Recordemos que la utilizamos para asignar un valor a una variable durante la ejecución de un programa. Esta instrucción se puede utilizar tanto para variables numéricas, como para alfanuméricas. Vamos a ver nuevas posibilidades sobre su utilización.

2.3.1 Entrada de varias variables

Vamos a suponer que necesitamos un programa que escriba el nombre de una persona, en un orden distinto del que se lo introducimos. Así, por ejemplo, si damos el nombre de JUAN GARCIA MARTINEZ, y deseamos que la máquina escriba GARCIA MARTINEZ, JUAN.

En primer lugar, deberemos preguntar cada palabra por separado. Utilizaremos la variable N$ para introducir el nombre, y las variables A$ y B$ para el primer y segundo apellidos, respectivamente.

Escribiremos entonces:

```
NEW
10 INPUT "NOMBRE: ";N$
20 INPUT "1ER. APELLIDO: ";A$
30 INPUT "2DO. APELLIDO: ";B$
40 PRINT A$;B$;N$
```

Al ejecutar el programa deberemos escribir las tres respuestas, en primer lugar el nombre, y pulsar la tecla de fin de línea. A continuación escribiremos el primer apellido, pulsaremos el fin de línea, y el segundo apellido, y otra vez fin de línea.

Veremos entonces en pantalla:

```
GARCIAMARTINEZJUAN
```

El programa ha escrito el nombre completo, pero tal como se lo hemos ordenado: una palabra a continuación de otra. Si queremos que nos deje espacios en blanco, o comas, se lo debemos indicar explícitamente. Así, deberemos escribir:

```
40 PRINT A$;" ";B$;", ";N$
```

Observe que tanto los espacios en blanco como la coma, han de formar parte del texto, se han de imprimir por tanto entre comillas.

Dado que debemos conocer el contenido de tres variables distintas, hemos utilizado tres instrucciones INPUT. Sin embargo, esto puede ha-

cerse de una forma más simple, utilizando una sola instrucción INPUT para las tres variables:

Una sola instrucción INPUT para varias variables

Una única instrucción INPUT puede utilizarse para introducir más de una variable.

Para ello, se escribe la palabra INPUT, seguida de la lista de variables a introducir, separadas por comas (,).

Así, podemos escribir el programa anterior utilizando únicamente dos instrucciones:

```
NEW
10 INPUT N$,A$,B$
20 PRINT A$;" ";B$;", ";N$
```

En general, al ejecutar un programa que contenga una instrucción INPUT con una lista de variables, la respuesta a esta instrucción deberá darse también en forma de lista. Sin embargo, para algunas máquinas, al efecto de una instrucción INPUT con una lista de variables es totalmente equivalente al de varias instrucciones INPUT. En todo caso, al ejecutar este programa en su microordenador podrá ver cómo funciona en él.

Así, al ejecutar el programa tal como lo tenemos ahora, para responder a la instrucción INPUT, deberemos escribir:

```
JUAN,MARTINEZ,GARCIA
```

¿Qué ocurre cuando no se contestan en número de variables pedidas?

De esta forma, cada texto será asignado a su respectiva variable, según el orden en que se haya escrito.

En pantalla veremos ahora:

```
MARTINEZ GARCIA, JUAN
```

El número de variables solicitadas, deberá evidentemente coincidir con el número de respuestas. Si el número de elementos de ambas listas (la lista de variables en la instrucción INPUT y la lista de valores que escribimos como respuesta) es diferente, pueden plantearse dos situaciones distintas:

1) Si el número de respuestas es mayor que el de variables que se preguntan, se asocian las primeras variables con las primeras respuestas y se ignoran las que se hayan introducido de más. Generalmente la máquina da un mensaje indicativo de esta situación.

2) La otra situación se produce si el número de elementos de la lista de respuesta es inferior al de variables preguntadas. En este caso la acción de la máquina puede diferir bastante, dependiendo del modelo. En algunos casos, la máquina da un mensaje de error, y detiene el proceso. En otros,

insiste en seguir preguntando, hasta que se haya completado la lista de variables solicitada.

Realice la prueba con su máquina, ejecutando el programa y escribiendo como respuesta únicamente dos palabras, por ejemplo: JUAN, MARTINEZ. Observe lo que hace la máquina.

Naturalmente, estas situaciones sólo se presentarán en los BASIC que admiten como entrada varias variables seguidas. En los otros tipos de BASIC en que esta posibilidad no existe no tendrá ningún problema de este tipo.

2.3.2 La cuestión de los separadores

En el programa que tenemos en memoria, utilizamos tres variables distintas, una para el nombre, otra para el primer apellido y otra para el segundo. Sin embargo, podríamos escribirlo de una vez, tal como queremos que nos lo imprima.

El programa se podría escribir ahora:

```
NEW
10 INPUT N$
20 PRINT N$
```

En este caso, deberemos contestar escribiendo directamente el nombre completo, es decir: MARTINEZ GARCIA, JUAN. Pero nos encontramos con un problema: en el texto incluimos un carácter, la coma (,) que tiene significado de separador. Por tanto, la máquina interpretará que hemos escrito dos textos distintos, el primero será MARTINEZ GARCIA, y el segundo será JUAN.

¿Cómo podemos entonces incluir una coma en el texto, sin que tenga significación de separador?

Las comillas permiten introducir textos que tienen coma

Esto puede hacerse escribiendo el texto, al dar la respuesta, entre comillas. De esta forma, la máquina toma exactamente lo escrito entre comillas, sin analizar el significado de ningún carácter, pudiendo de esta forma incluir tantas comas como se desee.

Comprobemos que esto es cierto, ejecutando el programa. Al contestar el nombre, deberemos escribir:

```
"MARTINEZ GARCIA, JUAN"
```

En pantalla aparecerá ahora:

```
MARTINEZ GARCIA, JUAN
```

Esto puede también utilizarse para incluir blancos al principio del texto.

Estos blancos normalmente son ignorados por la máquina. Comprobémoslo escribiendo el texto incluyendo los blancos que se quiera al principio: MARTINEZ GARCIA, JUAN.

Observaremos que en la pantalla no aparecen estos blancos. El texto se escribe justo al principio de la línea, los blancos son pues ignorados.

Para que no se supriman, debemos escribir todo el texto entre comillas. Así, volveremos a ejecutar el programa, y escribiremos como respuesta:

```
"          MARTINEZ GARCIA, JUAN"
```

observaremos que ahora la máquina lo escribe en la pantalla tal como lo hemos introducido, los blancos que hemos escrito quedan precediendo al texto.

De todas maneras, hay máquinas que suministran ya las comillas al pedir una variable textual, entonces el problema no se plantea. En este caso, las comas y los espacios en blanco se han escrito ya dentro de las comillas.

2.3.3 Instrucciones para dar la respuesta

Recordemos que la instrucción INPUT puede incluir un texto explicativo de lo que se pregunta. Este texto se escribe entre comillas, a continuación de la palabra INPUT, y seguido de una coma. Así, por ejemplo:

```
10 INPUT "NOMBRE",N$
```

Este texto, como ya hemos visto en repetidas ocasiones, aparece cuando se realiza la pregunta, de forma que nosotros podemos escribir la respuesta justo a continuación del mismo.

El texto informativo en un INDOT con varias variables

Esto puede ser de gran ayuda para utilizar programas que hacen varias preguntas —indicará qué es lo que está preguntando en cada caso—, o bien para dar una explicación sobre cómo hemos de escribir la respuesta. Así, si queremos escribir el nombre completo, incluida la coma, será conveniente indicarlo dentro de la misma pregunta. Escribiremos entonces:

```
10 INPUT "PONGA CON COMILLAS APELLIDO I
          APELLIDO II, NOMBRE",N$
```

Para practicar, volvamos de nuevo al programa de cálculo del descuento de los grandes almacenes. Tal como lo teníamos no daba ninguna indicación al ejecutarlo; simplemente hacía una pregunta y al final daba un número como resultado.

Podríamos mejorarlo escribiéndolo de esta forma:

```
NEW
10 INPUT "PRECIO ARTICULO: ",P : REM Introduccion del precio antiguo
20 LET D = 17.5 : REM Asignacion del valor del descuento
30 LET NP = P - P*D/100 : REM Calculo del precio rebajado
40 PRINT "PRECIO REBAJADO ";NP : REM Impresion del resultado
```

De esta forma al ejecutar el programa, nos indicará qué es lo que nos pregunta (PRECIO ARTICULO) y cuál es el resultado que nos da (PRECIO REBAJADO).

RESUMEN

La instrucción INPUT se utiliza para asignar el valor de una variable numérica o textual en un punto determinado del programa.

Además permite la múltiple asignación de valores a varias variables en una sola instrucción. A estas variables que se especifican en la instrucción INPUT se las denomina listas de variables.

Hay que distinguir entre la manera de escribir las variables en la instrucción y la manera de entrar los valores cuando la instrucción se ejecuta.

La escritura de la lista de variables se hace mediante la consecución de las variables, por el orden que deseemos que se entren y separadas mediante comas. La utilización de comas es universal, es decir, la permiten todos los lenguajes BASIC. Sin embargo, en algunos dialectos se puede utilizar el punto y coma.

En el momento que la instrucción se ejecuta dentro de un programa, el comportamiento consiste en pasar el programa y esperar a que se conteste a los valores de las variables.

La manera de contestar a estas preguntas es muy variada y depende de cada dialecto del BASIC. Sin embargo, se destacan dos modos que se deben tener en cuenta según el BASIC con que se trabaje.

— Las variables se deben entrar de una en una, es decir, el ordenador obliga a entrar una variable y a continuación la tecla de fin de línea, y entonces el ordenador pregunta la siguiente variable.

— Los valores de las variables se pueden entrar una detrás de otra separadas por comas y finalizar todo el paquete mediante una tecla de fin de línea. Cuando el ordenador obliga a entrar las variables de esta manera se pueden producir dos situaciones anómalas; que se entren más variables que las que realmente pide el ordenador, entonces el ordenador ignora las respuestas adicionales que se han dado, o bien que se entren menos valores de las que pide el programa. El comportamiento que sigue el ordenador es, o bien envía un mensaje de error o repite la pregunta para los valores que quedan por asignar.

En el caso de que el ordenador permita la entrada de diversas variables en una misma línea y separadas por coma, surge el pro-

blema de cómo entrar un valor textual que él mismo contenga una coma. La manera que el BASIC permite esta posibilidad es introducir unas comillas al inicio y al final del valor entrado como si fuera una constante textual.

Finalmente, aunque el INPUT utilice una lista de variables para entrar, se puede incluir al inicio una explicación de lo que se quiere entrar. Se escribe detrás de la instrucción INPUT el texto que se quiere dar como explicación encerrado entre comillas, como una variable textual y a continuación la lista de variables.

EJERCICIOS DE AUTOCOMPROBACION

Complete las frases siguientes:

21. La instrucción .... permite entrar datos para varias variables.

22. Al conjunto de variables que se entran en una única instrucción INPUT se las denomina .... de las variables.

23. En BASIC estándar se debe utilizar únicamente el separador ...., en otros dialectos se admite también el punto y coma.

24. En el momento de ejecutarse una instrucción INPUT con varias variables, en el .... estándar entran los diferentes valores separados por comas.

25. Debemos dar el mismo número de datos que .... contiene el INPUT.

26. El texto explicativo en la instrucción INPUT (SI/NO) .... puede utilizarse en el caso de que se tenga una lista de variables.

27. Como comportamiento general cuando se dan más datos de los que pide la instrucción INPUT el BASIC estándar los ....

28. Como comoportamiento más general cuando se dan menos datos de los que pide la instrucción INPUT al BASIC estándar da un ....

29. Cuando hay que entrar un texto con una coma es necesario encerrarlo entre .... para que no se confunda con el separador de diversos valores en una instrucción INPUT con varias variables.

30. En la lista de variables de una instrucción INPUT se pueden mezclar variables .... y textuales.

Encierre en un círculo las respuestas que correspondan a alternativa correcta.

31. La instrucción INPUT permite que se entren en la misma instrucción varias variables

   a) Numéricas.
   b) Textuales.
   c) Textuales y numéricas mezcladas.
   d) Primero las numéricas y luego las textuales.

32. Cuando una instrucción INPUT nos pide que entren varias variables se debe contestar

   a) Con menos de las que pide.
   b) Exactamente los datos que nos pide.
   c) Con más datos que los que pide.
   d) Se le indica primero el número de datos que se van a entrar.

33. A una instrucción INPUT con varias variables

   a) Se le puede añadir texto explicativo.
   b) No se le puede añadir texto explicativo.
   c) Debe tener un texto explicativo con un máximo de 5 letras.
   d) Tiene una constante numérica como texto explicativo.

34. En el BASIC estándar es posible responder, a un INPUT con varias variables, una lista de valores separados por:

   a) 10 espacios en blanco.
   b) 1 espacio en blanco.
   c) Un punto y coma.
   d) Una coma.

35. Cuando hay que entrar una lista de valores separados por comas, y un texto que a su vez contiene la coma, se entra el texto entre:

a) Paréntesis.

b) Comillas.

c) No se puede entrar.

d) Espacios en blanco.



2.4 VOLVER A EMPEZAR. INSTRUCCION GOTO

En todos los programas que hemos visto hasta ahora, las instrucciones se van ejecutando tal como aparecen en el listado, el proceso empieza por la instrucción que tiene el número más bajo, y finaliza por el que tiene el número más alto, y en ningún momento podemos alterar este orden.

Esto podemos comprobarlo con el siguiente programa:

```
10 PRINT "PRIMERA SENTENCIA"
20 PRINT "SEGUNDA"
30 PRINT "TERCERA"
40 PRINT "CUARTA"
50 PRINT "QUINTA Y ULTIMA"
```

Al ejecutar el programa, veremos en pantalla:

```
PRIMERA SENTENCIA
SEGUNDA
TERCERA
CUARTA
QUINTA Y ULTIMA
```

Sin embargo, existe la posibilidad de alterar este orden. Vamos a ver una instrucción que nos permite «saltar» de una sentencia a otra distinta de la siguiente. Esta instrucción se denomina GOTO.

GOTO significa en inglés «IR A», y se escribe seguida de un número. Este debe corresponder a un número de sentencia existente en el programa.

El GOTO significa salto a una instrucción

El efecto de esta instrucción será el salto a la sentencia cuyo número se indica.

Así, escribamos por ejemplo:

```
25 GOTO 50
```

Observemos ahora cómo se ha modificado la ejecución del programa. Escribiendo RUN, aparecerá en pantalla:

```
PRIMERA SENTENCIA
SEGUNDA
QUINTA Y ULTIMA
```

Hemos «saltado» —transferido el control— desde la sentencia número 25 a la sentencia número 50.

El flujo de un programa

El flujo del programa —orden en que se ejecutan las sentencias— ha dejado de ser lineal. El flujo del programa lineal, o sea sin esta nueva sentencia podemos observarlo en la figura 3. Al incluirla, observamos que se produce un salto, tal como se ve en la figura 4.

En este caso, hemos realizado un salto hacia una sentencia posterior. Puede saltarse también hacia sentencias anteriores, lo cual permite que se repitan —se vuelvan a ejecutar— una o más sentencias. En el apartado siguiente veremos las posibilidades que nos ofrece el salto hacia atrás: volver a empezar.

2.4.1 Lazos o bucles

Según lo estudiado, si queremos que una cosa se repita varias veces, debemos escribir la correspondiente instrucción tantas veces como sea necesario. Así, para escribir las palabras HOLA y ADIOS una debajo de otra tres veces, tendríamos que escribir:

```
NEW
10 PRINT "HOLA"
20 PRINT "ADIOS"
30 PRINT "HOLA"
40 PRINT "ADIOS"
50 PRINT "HOLA"
60 PRINT "ADIOS"
```

```
10  PRINT "PRIMERA SENTENCIA"
20  PRINT "SEGUNDA"
30  PRINT "TERCERA"
40  PRINT "CUARTA"
50  PRINT "QUINTA Y ULTIMA SENTENCIA"
```

Figura 3. Programa de flujo lineal

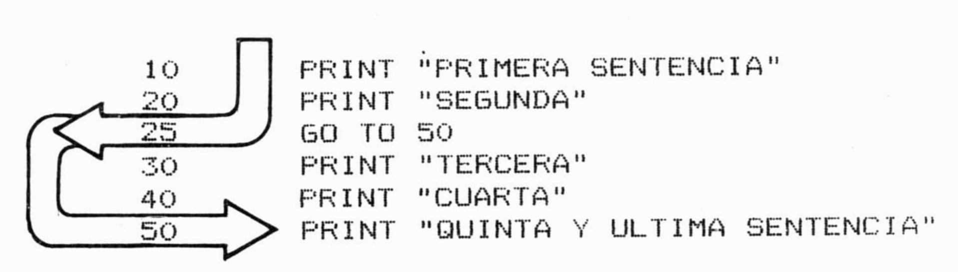


Figura 4. Programa con una bifurcación

Si queremos repetirlo diez veces debemos escribir 20 instrucciones, diez para cada palabra «HOLA» y diez más para cada palabra «ADIOS».

Esto no parece muy práctico. Otra forma de que se nos repita es escribirlo sólo una vez y ejecutar el programa tantas veces como haga falta. Así, para escribir seguidas las palabras HOLA y ADIOS, podemos escribir de nuevo:

```
NEW
10 PRINT "HOLA"
20 PRINT "ADIOS"
```

y ejecutar este programa tantas veces como queremos. El inconveniente en este caso es que las sucesivas escrituras aparecen interrumpidas.

Cada vez que escribimos RUN, el programa se ejecuta empezando por la primera sentencia, pasa a la siguiente y finaliza.

Pero ahora conocemos una instrucción que, dentro del mismo programa, nos permitirá saltar a cualquier punto del mismo. En este caso, nos interesaría saltar de nuevo al principio, una vez ejecutada la sentencia última. Para ello, escribiremos:

```
30 GOTO 10
```

comprobemos ahora que es lo que sucede, escribiendo RUN.

¿Qué observamos en pantalla?

Aparecen las palabras:

```
HOLA
ADIOS
HOLA
ADIOS
HOLA
ADIOS
...
```

repetidas continuamente, de arriba abajo de la pantalla.

Procesos sin fin

Este proceso no termina nunca. En algunas máquinas se detiene momentáneamente al llegar al final de la pantalla, aunque no se interrumpe. Por las «Prácticas con el microordenador», ya sabe cómo funciona en su caso.

Hemos desencadenado pues, un proceso sin fin.

Vamos a ver por qué. Recuerde que el listado que habíamos escrito era el siguiente:

```
10 PRINT "HOLA"
20 PRINT "ADIOS"
30 GOTO 10
```

La ejecución del programa se inicia en la sentencia 10, sigue en la 20, y tal como indica la 30, volvemos de nuevo a la 10, con lo cual hemos iniciado un proceso que tiene principio pero no final, a no ser que lo detengamos expresamente.

Una posibilidad sería desconectar la máquina. Sin embargo, ésta es una forma algo drástica de detener un proceso, que además haría que perdiésemos el programa. Hay otra forma —algo más suave— de interrumpir un proceso de este tipo, y se hace pulsando una tecla, denominada tecla de interrupción. En el teclado aparece generalmente con el nombre de BREAK (rotura, interrupción). En cualquier caso, en la parte de prácticas está indicado cómo funciona en su máquina.

Detención de los procesos sin fin

Así, la tecla de interrupción nos permite detener cualquier proceso que la máquina esté ejecutando.

Comprobemos que esto es cierto pulsándola ahora. Hemos conseguido detener el proceso, la situación está de nuevo bajo nuestro control. Si escribe ahora LIST podrá ver el listado del programa que estaba ejecutándose:

```
10 PRINT "HOLA"
20 PRINT "ADIOS"
30 GOTO 10
```

Al saltar del final al principio, con la sentencia 30, hemos generado un bucle. Imaginemos de nuevo el flujo de programa: será una línea que comienza en la sentencia 10, pasa a la 20, sigue en la 30 y ésta hace que salte de nuevo a la 10, y así hasta que interrumpamos el proceso. La línea de flujo de este proceso (Fig. 5) hace que se denomine lazo o bucle.

Tal como lo hemos escrito, el proceso tiene principio, pero no final. Más adelante veremos otras instrucciones que nos permitirán interrumpir desde dentro del propio programa —es decir, controlar— el proceso repetitivo.

2.4.2 Contador de vueltas

Cuando iniciamos un bucle, la primera necesidad que se hace evidente, es la de controlar el número de veces que se repite, es decir, el número de vueltas que damos.

Necesitamos pues un contador, una variable que se incremente en una unidad —se le sume 1— a cada vuelta que demos. Esta variable ha de valer 0 en el momento de empezar a contar.

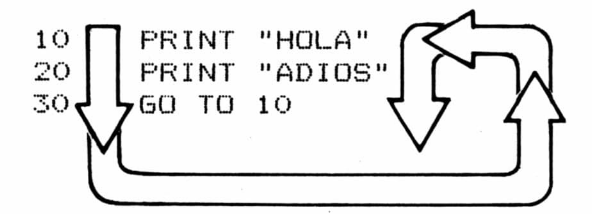


Figura 5. Programa con un bucle

Para incluir una variable que nos lleve la cuenta de las vueltas en nuestro programa, escribiremos estas dos sentencias:

```
 5 LET I=0
10 LET I=I+1
```

El listado del programa será ahora:

```
 5 LET I=0
10 PRINT "HOLA"
20 PRINT "ADIOS"
30 LET I=I+1
40 GOTO 10
```

Como incrementar una variable

Analicemos un poco estas dos sentencias que hemos añadido. En primer lugar hay que indicar al contador que no hemos dado ninguna vuelta. Dado que utilizamos la variable I para que nos lleve las cuentas, al empezar el proceso, antes de dar ninguna vuelta, esta variable deberá igualarse a 0, tal como hacemos en la sentencia número 5. Esto se ha de hacer antes de empezar el bucle, por esta razón, hemos incluido esta instrucción con un número inferior al de la sentencia 10, que es la primera del bucle.

Veamos ahora la sentencia número 30.

¿Cuál es el significado de la instrucción LET I = I+1? ¿Qué es lo que hace la máquina?

El efecto de esta instrucción es incrementar el valor de la variable I en una unidad. Veamos por qué.

Esta instrucción es de asignación. Sabemos que su función es asignar a la variable (I), situada a la izquierda del signo igual (=), el valor resultante de calcular la expresión de la derecha del mismo (I+1).

Lo que hace la máquina en primer lugar es calcular esta expresión:

— Toma el valor de la variable I, y le suma una unidad.

— Una vez hecho esto, el valor resultante es asignado a la variable situada a la izquierda del signo igual. En este caso, es la propia variable I.

De esta forma, cada vez que demos una vuelta, al pasar por la sentencia 30, el valor de I aumenta en uno. A la primera vuelta I valdrá 1 (resultado de sumar lo que valía, es decir 0, más la unidad) en la segunda vuelta, sumaremos 1 al valor de I, con lo que ahora valdrá 2, etc.

Comprobemos esto ejecutando el programa. Deje que se escriban unas cuantas veces las palabras HOLA y ADIOS. Interrumpamos ahora el proceso pulsando la tecla de interrupción correspondiente.

¿Cómo podemos saber cuántas vueltas hemos dado, es decir, cuántas veces se han escrito las palabras HOLA y ADIOS? La variable que lleva la cuenta es la variable I. Para conocer el valor de una variable, recuerde que podíamos utilizar la instrucción PRINT. Por tanto, escribiremos (sin numerar):

```
PRINT I
```

con lo que en pantalla aparecerá el valor de I. Este número, tal como hemos dicho, corresponderá al número de vueltas dado por el bucle.

Contadores

Esta es pues la forma de contar procesos que se repiten. Veremos que la utilización de un contador es necesaria en muchas ocasiones. Siempre se inicia fuera del bucle, y se va incrementando (de 1 en 1) dentro del mismo.



2.4.3 La instrucción de continuación

Sabemos cómo iniciar un proceso continuo —escribimos RUN—, y como detenerlo —pulsamos la tecla de interrupción—. Una vez parado podemos volver a empezarlo, escribiendo RUN de nuevo.

Sin embargo, existe la posibilidad de que el proceso siga a partir del momento en que lo hayamos interrumpido, sin necesidad de empezar de nuevo por el principio.

Para ello se escribe la palabra

```
CONT
```

que corresponde a la abreviatura de CONTINUE. Esta palabra se escribe igual en inglés y en castellano, y posee el mismo significado en ambas lenguas.

Seguimos en el punto donde habíamos detenido el programa

Vamos a utilizarla pues para seguir el proceso a partir del punto en que lo habíamos parado. Escribiremos pues (sin numerar) CONT.

Al pulsar el fin de línea, observaremos en pantalla como siempre las palabras HOLA y ADIOS repetidas. Si ahora interrumpimos de nuevo el proceso, podremos ver las vueltas que hemos dado escribiendo otra vez:

```
PRINT I
```

El valor que aparezca ahora en pantalla será el total de vueltas, las dadas antes de la primera interrupción, más las que hayamos dado al continuar el proceso. De esta forma pues, podremos interrumpir y proseguir el proceso tantas veces como se desee.

Atención:

No se deben modificar los programas si hay que utilizar el CONTINUE

Si desea continuar un proceso interrumpido, no puede modificar el programa. Así, no puede cambiarse, ni añadirse, ni borrarse ninguna sentencia del mismo.

Si desea realizar alguna modificación, podrá hacerlo, pero deberá recomenzar el proceso, escribiendo RUN. Es decir, en este caso es como si comenzase a ejecutar un nuevo programa.

2.4.4 Impresión del número de vueltas

A cada vuelta que da el bucle, se va incrementando el valor de la variable utilizada para contarlas. Hasta ahora, para conocer su valor, interrumpíamos el proceso e imprimíamos el valor de esta variable. Sin embargo, existe la posibilidad de incluir esta instrucción dentro del propio programa, de forma que nos contabilice el número de vueltas dadas, y además nos escriba este número en la pantalla. Para ello, escribiremos:

```
35 PRINT I
```

Si ejecutamos ahora el programa, veremos en pantalla:

```
HOLA
ADIOS
 1
HOLA
ADIOS
 2
...
```

Diferencia entre el RUN y el CONTINUE

Así, conocemos directamente el número de vueltas dadas. Interrumpamos ahora el proceso, y hagámoslo seguir, escribiendo CONT. Observaremos que el número no vuelve a empezar en 1, sino que continúa a partir del último valor que había escrito.

Si por el contrario lo interrumpimos y escribimos RUN a continuación, el proceso se inicia de nuevo, el contaje de las vueltas vuelve a comenzar desde 1.

2.4.5 Un ejemplo de utilización de un bucle

Recordemos una vez más el programa para calcular los precios rebajados de los artículos de unos grandes almacenes. Lo hemos ido documentando y perfeccionando, pero sigue teniendo un problema: cada vez que queremos calcular el precio de un artículo debemos dar la orden de ejecución, es decir, escribir RUN una vez por cada artículo.

Pero ahora hemos aprendido una instrucción que nos permite volver a empezar el proceso tantas veces como queramos. Añadiremos entonces una sentencia al programa, de forma que cuando haya finalizado un cálculo, vuelva al principio para preguntar otro precio, y así hasta que decidamos

no calcular más precios, momento en que deberemos pulsar la tecla de interrupción. Así, añadiremos la sentencia:

```
50 GOTO 10
```

con lo que el listado quedará:

```
10 INPUT "PRECIO ARTICULO: ",P : REM Introduccion del precio antiguo
20 LET D = 17.5 : REM Asignacion del valor del descuento
30 LET NP = P - P*D/100 : REM Calculo del precio rebajado lo
40 PRINT "PRECIO REBAJADO ";NP : REM Impresion del resultado
50 GOTO 10
```

Comprobemos el funcionamiento ahora, escribiendo RUN. El programa nos preguntará el precio actual y nos imprimirá el valor del precio rebajado de tantos artículos como queramos, hasta que nosotros pulsemos la tecla de interrupción. De esta forma, podremos calcular los precios de todos los artículos, escribiendo RUN una sola vez.

2.5 LA SENTENCIA STOP

Para ejecutar un programa sabemos que se utiliza la palabra RUN. Esta palabra es la orden que desencadena la ejecución del proceso, empezando por el principio.

Supongamos ahora que tenemos el siguiente programa:

```
NEW
10 PRINT "UNO"
20 PRINT "DOS"
30 PRINT "TRES"
40 PRINT "CUATRO"
50 PRINT "CINCO"
```

Si escribimos RUN para ejecutarlo veremos en pantalla:

```
UNO
DOS
TRES
CUATRO
CINCO
```

Siempre que escribamos RUN, ejecutará en primer lugar la primera

Empezar un programa en una instrucción que no sea la primera

sentencia, en este caso la número 10. Sin embargo, podemos alterar esto indicando el punto del programa por el que deseamos que empiece, caso de que ésta no sea la primera sentencia. Para ello, escribiremos la palabra RUN, seguida de un número de sentencia. Por ejemplo:

```
RUN 30
```

Pulsando ahora el fin de línea veremos en pantalla:

```
TRES
CUATRO
CINCO
```

En esta ocasión, el programa se ha empezado a ejecutar por la sentencia 30.

Así, podremos indicar el número de la sentencia por la que queremos que empiece el proceso. La máquina la ejecutará en primer lugar, y seguirá por las que estén escritas a continuación, ignorando totalmente las sentencias anteriores.

Esto nos puede servir para ejecutar un trozo de programa. siempre que esté situado al final del mismo.

Sin embargo, si quisiéramos que se ejecutara el primer trozo únicamente, no sabemos cómo hacerlo, ya que una vez desencadenada la ejecución, la máquina sigue hasta el final.

El STOP, la detención programada de un proceso

Vamos a ver que existe una instrucción que nos permite detenernos en un punto determinado. Es la instrucción STOP. Esta palabra significa «ALTO», tiene el mismo significado que la señal del código de circulación: deténgase.

Para incluirla en el programa, escribiremos:

```
25 STOP
```

Si lo ejecutamos ahora, el programa escribirá en pantalla

```
UNO
DOS
```

deteniéndose inmediatamente.

Si ahora quisiéramos que continuara el proceso podríamos utilizar una instrucción que ya hemos visto anteriormente: la instrucción CONT.

Tal como habíamos visto la palabra CONT es la abreviatura de CONTINUE, y la utilizábamos para continuar un proceso interrumpido mediante la tecla de interrupción.

De la misma manera, si hemos detenido el programa en un punto de-

terminado por medio de una instrucción STOP, podemos hacer que prosiga a partir de este punto, escribiendo directamente CONT.

En este caso veremos en pantalla que se sigue con las palabras:

```
TRES
CUATRO
CINCO
```

Recuerde que el UNO y el DOS ya estarán en pantalla y estas últimas líneas aparecerán después.

La diferencia entre utilizar la tecla de interrupción o la instrucción STOP —dentro del programa— para interrumpirlo, estriba en el hecho de que en un caso podemos precisar el punto en que debe detenerse y en el otro no.

En el programa que tenemos en memoria, aseguramos que el proceso se interrumpe entre las sentencias 20 y 30, ya que hemos situado la instrucción STOP entre ambas (le hemos dado el número 25). Si por el contrario detenemos el proceso con la tecla de interrupción, no podemos asegurar el punto en que se detendrá, ya que la máquina ejecuta las sentencias a gran velocidad, de forma que resulta prácticamente imposible acertar a pulsar la tecla justo en el momento preciso.

De esta forma con la instrucción RUN numerada, la instrucción STOP situada convenientemente, y la instrucción CONT, podremos ejecutar trozos de programa, independientemente del resto del mismo.

De esta manera, podemos hacer pruebas de cada parte de un programa, asegurándonos de que funcionan tal como nosotros queremos.

RESUMEN

La instrucción GOTO (IR A) permite cambiar el orden de ejecución de las sentencias de un programa.

Detrás de la palabra GOTO se indica el número de instrucciones a la que se desea saltar. En lenguaje informático este salto de instrucciones se denomina transferencia del control.

El orden con que se ejecutan las sentencias de los programas se denomina flujo del programa. Cuando el orden numérico de las instrucciones coincide con el orden real de ejecución el programa se dice que tiene un flujo lineal. La instrucción GOTO permite cambiar el flujo de ejecución.

Uno de los usos de la instrucción GOTO es formar lazos o bucles, de tal manera que se pueden repetir unas determinadas instrucciones muchas veces. Tantas que el proceso puede no tener fin.

Estos procesos sin fin a veces son planeados por los programadores, otras veces se deben a errores del programador. En cualquier caso es necesario poderlos interrumpir. Para ello se dispone de la tecla de interrupción que permite detener la ejecución del programa en el punto que se encuentre en aquel momento.

Sin embargo, después de apretar la tecla de interrupción, nos damos cuenta de que deseamos continuar el proceso en el lugar donde se ha interrumpido. El BASIC dispone del comando CONTINUE para proseguir el proceso que se había interrumpido.

Entre la tecla de interrupción y continuación se pueden visualizar todas las variables que el programa hace funcionar mediante la instrucción PRINT inmediata; pero si se desea continuar el programa con el comando CONTINUE, no se debe alterar ninguna sentencia.

Si es imprescindible modificar una instrucción, es necesario volver a empezar el proceso mediante el comando RUN.

La mayoría de programas pueden pensarse como bucles de este estilo; cuando se han acabado las instrucciones una instrucción GOTO, permite volver a empezar. Hay que tener en cuenta que este volver a empezar difiere en el comportamiento del volver a empezar con el comando RUN.

Cuando volvemos a empezar con la instrucción GOTO, las variables quedan con los valores que tenían después de la última sentencia del programa. En cambio, el comando RUN las deja todas indefinidas, es decir, sin un valor específico.

El BASIC permite mediante la instrucción STOP parar el programa en cualquier parte del mismo. El comportamiento es idéntico a la tecla de interrupción. La diferencia consiste en que, con la tecla de interrupción, no sabemos dónde paramos el programa exactamente.

Finalmente, la tecla RUN puede ir seguida de un número para indicar que la ejecución se inicie por la sentencia que indica este número, en lugar de la sentencia con numeración más baja en el programa.

EJERCICIOS DE AUTOCOMPROBACION

Encierre en un círculo las respuestas que correspondan a la alternativa correcta.

36. La instrucción GOTO transfiere el control de ejecución, es decir, la sentencia siguiente que se ejecutará es la:

   a) Que resulta de sumar el número de sentencia que tiene el GOTO con el número que sigue a la palabra GOTO.

   b) Que sigue a la palabra GOTO.

   c) La siguiente a la que sigue a la palabra GOTO.

   d) Que resulta de restar el número de sentencia que tiene el GOTO con el número que sigue a la palabra GOTO.

37. La tecla de interrupción se utiliza para

a) Detener la ejecución del programa en la sentencia que se desea.

b) Para el programa cuando se desea, sin saber exactamente en la instrucción que se detendrá.

c) Después de transcurridas 5 instrucciones más.

d) Después de transcurridos 10 segundos.

38. La instrucción STOP se utiliza para detener la ejecución del programa:

a) Hora de tener la ejecución del programa en la sentencia que se desea.

b) Para parar el programa cuando se desea sin saber exactamente en la instrucción que se detendrá.

c) Después de transcurridas 5 instrucciones más.

d) Después de transcurridos 10 segundos.

39. Para continuar un programa que hemos detenido se utiliza la instrucción CONTINUE. Para utilizarla correctamente.

a) No se puede modificar el programa.

b) Se debe modificar el programa.

c) Antes hay que ejecutar el comando RUN.

d) Debe utilizarse después de la parada mediante la instrucción STOP.

40. La instrucción RUN se puede complementar con un número detrás. Este número indica

a) Las veces que debe repetirse el programa.

b) Que se detenga cuando el valor de una variable numérica coincida con este número.

c) Que el programa tiene para ejecutarse este número de segundos.

d) Que inicie la ejecución en la sentencia que lleva este número.

En las instrucciones siguientes, rodee con un círculo la V si la instrucción es correcta y la F si es falsa.

41. REM PRINT Ya he llegado V F

42. INPUT A$: REM Entrar el nombre V F

43. PRINT TAB(10), A$ V F

44. GOTO -23 V F

45. INPUT «Entre las coordenadas», x,y V F

46. PRINT «INPUT»; «Es una instrucción», «de BASIC» V F

47. RUN CONTINUE: PRINT REM V F

48. RUN -50 V F

49. PRINT «A»;«es la primera letra del abecedario»; :INPUT A V F



50. INPUT A$,B$,C$: REM Entrada del nombre y apellidos V F

Capítulo 3

• Expresiones aritméticas y textuales.
• Representación interna de los datos.
• Estudio de las funciones.

ESQUEMA DE CONTENIDO

Objetivos	
Nota sobre la instrucción LET	
Expresiones aritméticas	Operadores binarios y unarios Potenciación Operaciones consecutivas Jerarquía de los operadores
Representación interna de los datos	Sistema de numeración Sistema binario Agrupaciones de bits
Expresiones textuales	El código ASCII El operador de concatenación Extracción de segmentos de texto
Casos particulares de las expresiones	Números muy grandes Números muy pequeños Textos demasiado largos
Funciones intrínsecas	
Funciones numéricas	Catálogo de funciones numéricas Ejemplos de utilización
Funciones numéricas y argumento textual	Catálogo de funciones Ejemplos de utilización
Funciones textuales	Catálogo de funciones Ejemplos de utilización
Funciones textuales y argumento numérico	Catálogo de funciones Ejemplos de utilización
Control de pantalla	Borrado de pantalla Control de la posición

3.0 OBJETIVOS

El primer objetivo de este capitulo es aprender a manejar las expresiones tanto aritméticas como textuales. Respecto a las expresiones aritméticas aprenderemos el concepto de operador unario y binario. También aprenderemos el orden de prioridad con que se efectúan las operaciones en una expresión compleja. Por otra parte, veremos los problemas que pueden surguir al manejar expresiones con números muy pequeños o muy grandes. Respecto a las expresiones textuales aprenderemos cómo unir o separar trozos de textos.

El segundo objetivo es conocer la presentación interna de los datos en el ordenador. Para ello estudiaremos el sistema binario y cómo realizar las conversiones entre sistema binario y sistema decimal. Veremos también cómo se almacenan los datos textuales codificando cada símbolo en el sistema binario según el código ASCII. Entender la estructura interna del ordenador nos facilitará la comprensión de su funcionamiento.

Por último estudiaremos qué son las funciones y cómo se manejan. Las funciones, de las que el BASIC tiene un amplio repertorio, constituyen una herramienta muy potente para realizar todo tipo de operaciones aritméticas, trigonométricas, textuales, etc. Un tipo especial de funciones las utilizaremos para controlar la posición de escritura en la pantalla.

3.1 NOTA SOBRE LA INSTRUCCION LET

Como ya sabemos, en BASIC las instrucciones empiezan por un verbo o palabra clave. Esta palabra clave es la que indica la acción que debe efectuarse sobre los complementos. Sin embargo, en el caso particular de la instrucción de asignación de un valor a una variable, esta indicación es algo superflua. En efecto, la instrucción

```
LET Y=100
```

se lee «sea Y igual a 100», pero también se entendería perfectamente si se leyera «Y igual a 100» omitiendo la palabra «sea». Por esta razón, algunas variaciones del BASIC permiten escribir la instrucción de asignación suprimiendo la palabra LET. En el ejemplo anterior sería

```
Y=100
```

que es totalmente equivalente.

En algunos BASIC se puede prescindir del verbo LET en una instrucción de asignación

En las «Prácticas con el microordenador» ya veremos si es posible utilizar esta simplificación en nuestro ordenador.

Debe tenerse en cuenta que en todas las variantes del BASIC se admite la instrucción de asignación escrita de forma completa, es decir, con la palabra LET delante. Por esta razón, en este curso utilizaremos siste-

máticamente la forma completa. No obstante, si consultamos listados de programas realizados con otros ordenadores, debemos tener presente que en ellos es posible que no aparezca LET al efectuar asignaciones. Asi mismo, si nuestro ordenador admite la forma simplificada, los ejemplos de este texto que están en forma completa se podrán escribir en forma simplificada sin que ello afecte a su funcionamiento.



3.2 EXPRESIONES ARITMETICAS

Antes de empezar debemos hacer una advertencia previa. Es posible que al realizar operaciones complicadas no obtengamos resultados absolutamente exactos. Así, puede ocurrir que en lugar de obtener, por ejemplo, un 4 de resultado, obtengamos un 3.99999. Esto es inevitable ya que los ordenadores trabajan con el sistema binario con un número limitado de cifras y por tanto se va acumulando un pequeño error a cada operación.

A veces los resultados no son totalmente exactos

En el capítulo anterior se ha descrito la instrucción PRINT que permite consultar el contenido de las memorias o variables que se han definido previamente. El punto que se considera seguidamente es la realización de cálculos más complejos con las variantes o constantes.

3.2.1 Operadores binarios y unarios

El cálculo más simple es realizar una suma. Por ejemplo, si escribimos:

```
PRINT 3+4
```

En la pantalla aparece un 7 que corresponde al resultado de la operación. El símbolo «+» indica que se realiza una operación (en este caso la suma) y por esta razón se le denomina operador. En general, los operadores son símbolos que se colocan entre variables o constantes para realizar las operaciones. Los más corrientes son los operadores aritméticos que se refieren a las operaciones aritméticas y son la suma, la resta, la multiplicación, la división y la potenciación. En BASIC los símbolos que se utilizan son:

+ Suma

− Resta

* Multiplicación

/ División

^ Potenciación

Nótese que para el signo de multiplicar no se puede utilizar la letra «x» puesto que se confunde con la variable del mismo nombre. En su lugar se usa, como ya sabemos, el símbolo «*» que se denomina asterisco. Como

ejemplo práctico puede escribirse el siguiente programa que sirve para imprimir las tablas de multiplicar.

```
 10 INPUT A
 20 PRINT 0*A
 30 PRINT 1*A
 40 PRINT 2*A
 50 PRINT 3*A
 60 PRINT 4*A
 70 PRINT 5*A
 80 PRINT 6*A
 90 PRINT 7*A
100 PRINT 8*A
110 PRINT 9*A
120 PRINT 10*A
```

En la línea 10, el operador entra el número cuya tabla de multiplicar se desea. Las líneas entre 20 y 120 imprimen la tabla.

Pulsemos la tecla RUN y contestemos, por ejemplo, el 3. El programa nos dará la tabla del 3 completa.

Otros ejemplos de utilización de operadores:

```
PRINT 100-1     imprime  99
PRINT 10/5      imprime   2
```

En estas instrucciones no es necesario introducir ninguna variable. Observemos ahora dos instrucciones siguientes:

```
PRINT -3        escribira  -3
PRINT +3        escribira   3
```

En este caso el operador sólo afecta a una constante en lugar de dos como antes. Esto no es siempre posible. Por ejemplo, si se escribe:

```
PRINT *3   o bien   PRINT /3
```

Operadores mantener el signo y cambio de signo

el BASIC da un mensaje de error. Parece claro, pues, que existen dos tipos de operadores: unos que afectan a 1 ó 2 variables (o constantes) y otros que afectan a 2 variables o constantes. Los operadores que afectan a un sólo operando se denominan UNARIOS y los operadores que afectan a dos operandos se denominan BINARIOS. Los signos más (+) y menos (−) actúan de operadores unarios o binarios. Por el contrario los signos asterisco (*) y barra (/) son siempre operadores binarios. Cuando el signo menos (−) es de tipo unario, efectúa la operación de «cambio de signo». Si actúa como

binario, realiza la resta. Si el signo más (+) actúa como unario, efectúa la operación de «mantener el signo», es decir, que en realidad no hace nada puesto que los números se suponen positivos en principio. El conjunto operadores-operandos se denomina *EXPRESION*. En los apartados siguientes se ampliará el concepto de expresión.

Para que comprenda bien la diferencia entre el operador unario y el operador binario consideremos el programa siguiente:

```
10 INPUT "Primer Numero",A
20 INPUT "Segundo Numero",B
30 CLS
40 PRINT +A,+B
50 PRINT -A,-B
60 PRINT A+B,A-B
70 GOTO 10
```

Este programa escribe los resultados de la actuación del operador + como operador unario, es decir, mantener el signo, en la línea 40. En la sentencia 50 se escribe la actuación de los operadores unarios sobre los números entrados.

En la línea 60 se calculan los operadores + y − como binarios, es decir, como suma y resta aritméticas.

La sentencia 30 sirve para borrar la pantalla.

Ejecutemos el programa entrando el comando RUN y la tecla ENTER.

Como respuesta a la entrada del primer número respondamos un 3 y para el segundo respondamos un 4.

En la pantalla aparecerá:

```
 3     4
-3    -4
 7    -1
```

En la primera línea los valores de A y de B aparecen inalterados, pues la operación de mantener el signo no altera el valor de la variante.

En la segunda línea aparecen los valores de A y de B cambiados de signo, pues sobre ellos se ha aplicado el operador de cambio de signo.

Finalmente, en la última línea aparece la suma y la resta de A y de B. Observe que el resultado de la resta es negativo pues el valor del minuendo (A) es menor que el del sustraendo. Es comparable a si tengo 3 y me piden 4, quedo a deber 1, es decir, tengo menos 1.

Si en la siguiente pregunta del ordenador se entran los números −3 y −4 para las variables A y B. Es decir, se entran números negativos, en la pantalla aparecerá:

```
-3    -4
 3     4
-7     1
```

En la primera línea aparecen los números tal como se han entrado, ya que la operación mantener el signo no cambia los valores de las variables. En la segunda línea aparecen los valores A y B cambiados de signo, pues sobre ellos se ha aplicado el operador de cambios de signo. En la última línea la suma de los dos valores entrados es −7, si debía tres y le añado una deuda de 4 el resultado es una deuda de 7, es decir, tengo −7.

Finalmente, en la operación de la resta, el resultado es 1, ya que si debo 3 a un tendero y a esta deuda le quito la deuda de cuatro que el tendero tiene conmigo, en definitiva recibo 1.

3.2.2 Potenciación

Un signo nuevo que aparece en la lista de operadores es el «^».

Este símbolo se usa para indicar la operación de potenciación («elevado a»). Dependiendo del tipo de teclado e incluso del tipo de impresora utilizados, el signo de la potenciación será una flecha vertical (↑) o el acento circunflejo (ˆ).

Recordemos que elevar un número a una potencia significa que dicho número se multiplicará por sí mismo tantas veces como indique la potencia. Así, dos elevado a la cuarta potencia (2^4) significa $2 \times 2 \times 2 \times 2 = 16$. El número 2 se denomina base y el 4 se denomina exponente. Si se eleva un número a la segunda potencia, es decir que el exponente es un 2, entonces se dice que se eleva un número al cuadrado. Si el exponente es 3, se dice que se eleva al cubo. A partir de 4 se sigue la nomenclatura normal; es decir, elevado a la cuarta potencia o elevado a 4 simplemente, etc.

En BASIC, para indicar la operación de potenciación se escribe primero la base, luego la flecha (↑) o acento circunflejo (ˆ) y finalmente el exponente.

Veamos algunos ejemplos:

```
PRINT 4^2 vale 16   (4x4)
PRINT 2^3 vale  8   (2x2x2)
PRINT 3^4 vale 81   (3x3x3x3)
```

El BASIC no admite que la base sea negativa

En estos ejemplos hemos contemplado los casos más sencillos puesto que tanto la base como el exponente eran números positivos. ¿Qué ocurre si ello no es así? En primer lugar, aunque en matemáticas se admiten en ciertos casos las bases negativas, en BASIC no son válidas nunca. Por el contrario, sí que se admite que el exponente sea positivo, cero o negativo.

Exponente negativo en la potenciación

Las matemáticas nos dicen que elevar un número a un exponente negativo equivale a calcular la potencia y a continuación evaluar un inverso. Es decir que

$$2^{-3} \text{ equivale a } \frac{1}{2^3}$$

Por tanto, si escribimos las dos instrucciones siguientes

```
10 LET Y=-3
20 PRINT 2^Y
```

estamos elevando el número 2 a la menos tres, puesto que el valor Y es −3. Esto equivale a dos elevado al cubo que es 8 y a continuación se calcula el inverso; es decir, $\frac{1}{8}$, que es 0.125, tal como aparece en pantalla.

Exponente nulo en potenciación

Por convención, cualquier base elevada a un exponente cero da como resultado el número 1. Escribiendo la instrucción

```
PRINT 2^0.75^0
```

vemos que los dos resultados (dos elevado a cero y setenta y cinco elevado a cero) valen 1.

Nos queda un último caso. Hemos visto al principio los exponentes positivos, pero, ¿qué sucede si el exponente es fraccionario (con decimales)? Por ejemplo, dieciséis elevado a un medio. Escribamos

```
PRINT 16^0.5
```

Exponente fraccionario en la potenciación

El resultado es 4. De nuevo, las matemáticas nos informan que elevar un número a una fracción equivale a calcular la raíz indicada en la fracción. Es decir que

$$16^{1/2} \text{ equivale a } \sqrt{16}$$

Según esto, deiciséis elevado a un medio equivale a la raíz cuadrada de 16, o sea 4. Si calculamos 8 elevado a un tercio, obtendremos un 2 ya que la raíz cúbica de 8 es 2. En efecto si escribimos

```
LET Y=1/3
PRINT 8^Y
```

el resultado es 2. Tenemos, pues, un medio muy potente para calcular raíces cuadradas, cúbicas, cuartas, etc.

En este punto hay que hacer algunas consideraciones. Los dos casos más frecuentes en los que utilizaremos exponentes será para calcular cuadrados y raíces cuadradas. A nosotros, para calcular el cuadrado de X, nos da igual escribir X^2, o bien X*X. No obstante, para la máquina es mucho más rápido el cálculo de X*X que el de X^2. Dada la extraordinaria rapidez de cálculo de los ordenadores, esta diferencia sólo se manifiesta si este cálculo está en una zona del programa que se efectúa varias veces durante un proceso. Sin embargo, como costumbre recomendable, escribiremos X*X en lugar de X^2.

La función SQR es otro método más rápido para sacar la raíz cuadrada

Un caso análogo ocurre con las raíces cuadradas. Para calcular la raíz cuadrada de X, se escribe X^0.5. Sin embargo, puesto que esta operación es muy frecuente, se ha diseñado un sistema para evaluar una raíz cuadrada de forma más rápida. Para ello se usa la palabra clave SQR que proviene de «square root» (raíz cuadrada en inglés). Si queremos calcular la raíz cuadrada de 16 escribimos

```
PRINT SQR(16)
```

que da un 4 como resultado. Observamos el hecho de que el número sobre el cual se aplica la operación se escribe entre paréntesis a la derecha de la palabra SQR. Las operaciones que se realiza mediante la utilización de palabras clave como operadores, se denominan funciones. El BASIC dispone de un buen repertorio de funciones que serán descritas más adelante.

3.2.3 Operaciones consecutivas

Hasta ahora se ha visto cómo podemos realizar operaciones muy elementales. ¿Qué hacer si el problema es más complicado? Por ejemplo, se compran 4 botellas a un precio de 500 cada una, y 3 paquetes con un precio unitario de 400. Cuál es el valor total de la compra:

(4*500) + (3*400)

Los resultados intermedios se cierran entre paréntesis

Aquí aparecen unos símbolos nuevos: los paréntesis de apertura «(» y de cierre «)». En la forma normal de expresar los cálculos, los paréntesis indican que se calculan dos resultados intermedios, 4*500 y 3*400, y luego efectúan la suma. Pues bien, en BASIC, el significado de los paréntesis es el mismo, es decir, indican que las operaciones englobadas constituyen un resultado intermedio que debe evaluarse previamente. Por tanto, para resolver el problema de la compra escribiremos el programa siguiente:

```
10 LET B=4
20 LET P=3
30 PRINT (B*500)+(P*400)
```

La variable B contiene el número de botellas adquiridas y la variable P el número de paquetes. En realidad, el BASIC considera a los paréntesis como un operador que sirve para que la expresión que se encuentra en su interior se evalúe y se tome como resultado intermedio. Por esta razón es posible aplicar los operadores unarios (−, +) sobre una expresión entre paréntesis.

Ejemplo:

```
PRINT -(3+4)
```

El resultado será -7.

Como ya sabemos, la instrucción LET se utiliza para asignar valores a las variables. Pero ésta no es la única aplicación de esta sentencia. Podemos usarla también para evaluar expresiones y asignar el resultado a la variable.

Ejemplo:

```
LET A=3+4
```

El resultado de la expresión es 7 y se asigna a la variable A.

3.2.4 Jerarquía de los operadores

Para cálculos más complicados se combinan varios operadores y operandos hasta formar expresiones muy extensas. La longitud de una expresión está limitada únicamente por la longitud de la línea de programa BASIC que es de 255 caracteres.

Ejemplo:

Programa para calcular el área de un triángulo.
El área de un triángulo es igual al producto de
la base por la altura dividido por 2.

```
10 INPUT "Base:",B
20 INPUT "Altura:",A
30 LET S=B*A/2
40 PRINT "Area=";S
```

En este ejemplo, la expresión B*A/2 puede interpretarse como:

$$B * \frac{A}{2} \quad \text{o} \quad \frac{B*A}{2}$$

Es indiferente considerar uno u otro caso, puesto que el resultado es idéntico en ambos. Esto no siempre es así y existen una serie de normas para evaluar una expresión. Por ejemplo, la expresión 3+4*2, ¿debe interpretarse como (3+4) * 2 cuyo resultado es 14, o bien como 3 + (4*2) cuyo resultado es 11? Para responder a esta pregunta podemos conocer las normas de evaluación de las expresiones. Estas normas son:

Normas para la evaluación de las expresiones

a) La expresión se evalúa de izquierda a derecha.

b) Cada operador tiene un nivel jerárquico.

c) Si a la derecha de un operador se encuentra un operador de menor nivel, se evalúa previamente el de menor nivel.

Operador	Jerarquía
( )	0
^	1
+ (Unario)	2
− (Unario)	2
*	3
/	3
+ (Suma)	4
− (Resta)	4

Figura 1 Jerarquía de los operadores

En la tabla de la figura 1 se detallan los niveles jerárquicos de los operadores.

Según esta tabla, la expresión 3+4*2 se evaluará de la siguiente manera: El primer operador que se encuentra empezando por la izquierda es el signo más (+) que tiene un nivel jerárquico de 4. Se mira entonces cuál es el siguiente operador.

Este es el signo de multiplicación (*) que tiene un nivel de 3. Puesto que es de nivel inferior se efectúa primero la multiplicación y a continuación se realiza la suma. Por tanto el resultado es 11. Este ejemplo se presenta gráficamente en la figura 2.

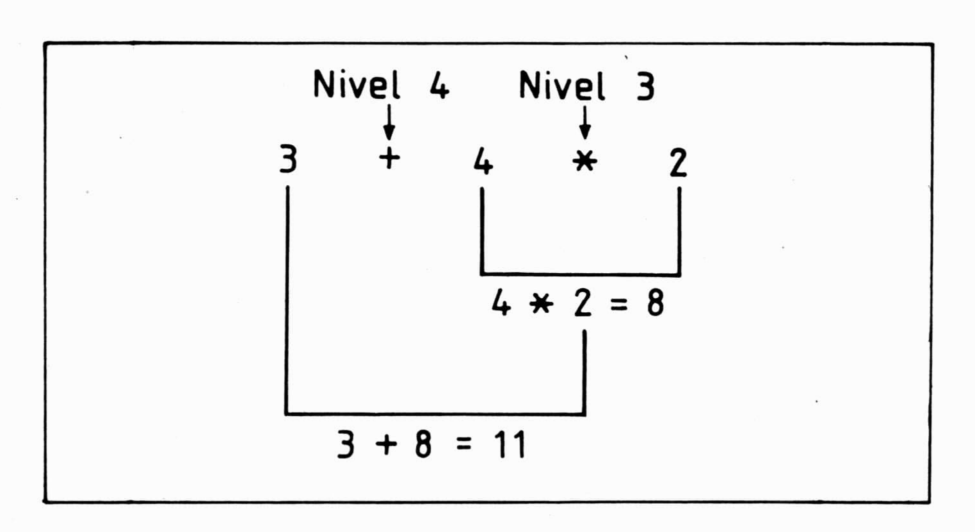


Figura 2 Representación gráfica de la jerarquía de los operadores en una expresión

Los paréntesis tienen el nivel jerárquico más bajo. Esta característica permite usarlos para cambiar el orden de prioridad de un operador. Tomemos como ejemplo la expresión 12/2*3. Siguiendo las reglas enunciadas anteriormente se evalúa de la siguiente manera: El operador de división (/) tiene un nivel de 3. A su derecha se encuentra el asterisco (*) que tiene el mismo nivel. Por tanto se efectúa la división, según la norma a), es decir,

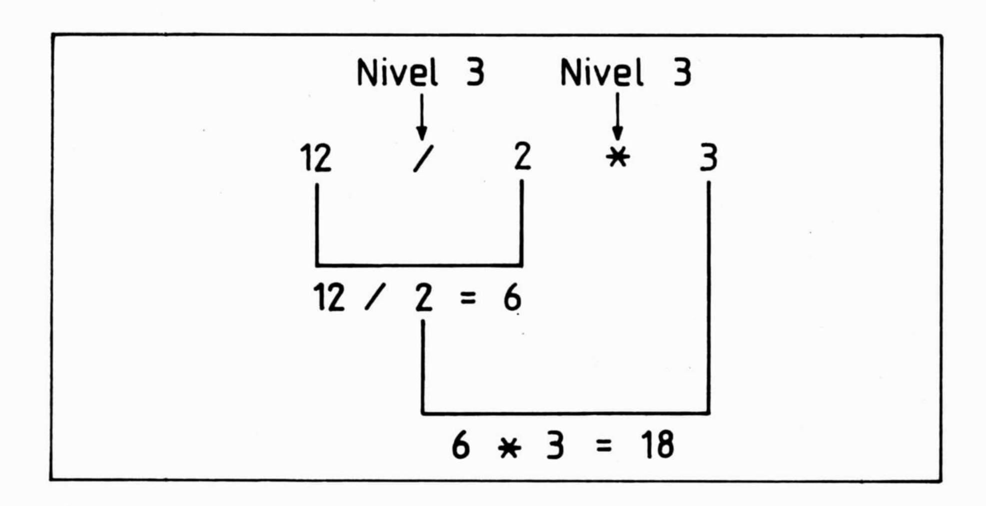


Figura 3 Representación gráfica de la jerarquía de los operadores en una expresión

de izquierda a derecha, y el resultado se multiplica por 3. En la figura 3 se representa cómo se efectúa esta operación.

Si lo que en realidad deseamos es dividir el número 12 por el resultado de multiplicar 2 por 3 hay que cambiar el orden de prioridad. Esto se realiza utilizando los paréntesis. Es decir, escribimos:

12/(2*3)

El paréntesis cambia el orden de prioridad de ejecución

Al evaluar la expresión (inténtelo antes de ver la figura) se obtiene lo que muestra la figura 4

Entonces el BASIC encuentra el signo de división con nivel 3. A su derecha encontrará el símbolo «(» que tiene nivel cero. Por tanto, se efectúa la operación correspondiente al paréntesis. Esta operación es el «cambio

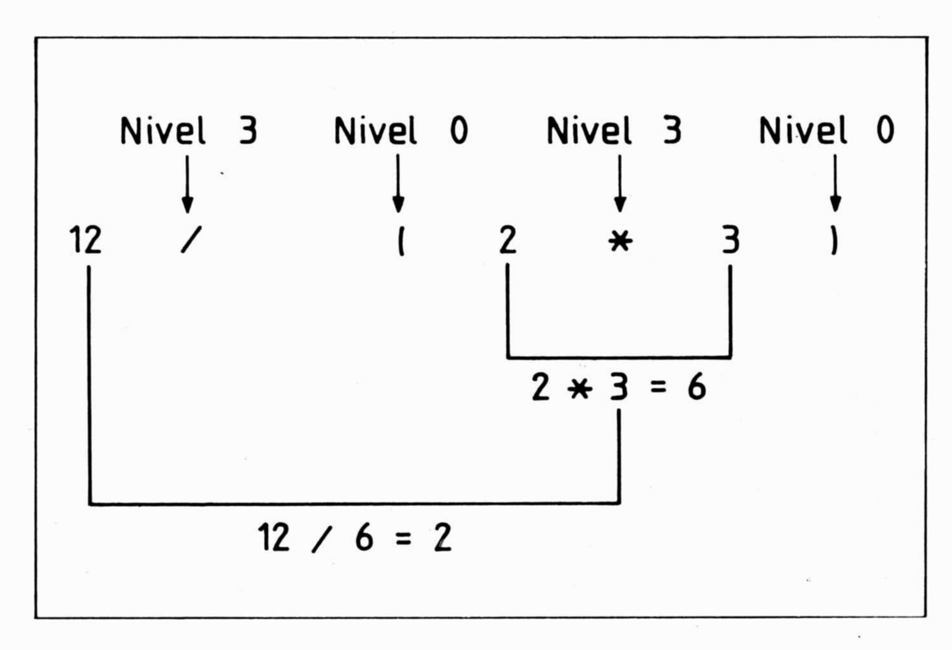


Figura 4 Representación gráfica de la jerarquía de los operadores en una expresión

de prioridad en la ejecución» y no afecta a los operandos. A continuación se encuentra el signo de multiplicación. Se mira entonces a su derecha donde aparece el paréntesis de cierre «)» con nivel cero. Se efectúa la operación asociada a este símbolo. Esta operación es el «fin cambio de prioridad» y tampoco afecta a los operandos. Puesto que ya no hay más operandos, se realizan las operaciones pendientes. La última de ellas era la multiplicación. Se obtiene el valor 2*3=6. Finalmente se efectúa la división y el resultado es 12/6=2.

Los paréntesis deben estar balanceados dentro de una expresión

Una expresión entre paréntesis puede estar englobada dentro de otra que también esté entre paréntesis. La única condición es que los paréntesis estén BALANCEADOS. Esto significa que por cada paréntesis de apertura exista un paréntesis de cierre. No hay limitación, en principio, en el número de paréntesis interiores que se quieran utilizar en una expresión. Obviamente, existe siempre el límite de la longitud total de la línea de programa que no puede sobrepasar el valor de 255 caracteres. Vea en la figura 5 cómo se evalúa la siguiente expresión:

2 * (1+8 / (3+1))

La misión fundamental de los paréntesis es cambiar el orden de prioridad en la evaluación de una expresión. Se emplean también a menudo para dar mayor claridad a las expresiones complejas aunque no sean estrictamente necesarios.

Ejemplo:



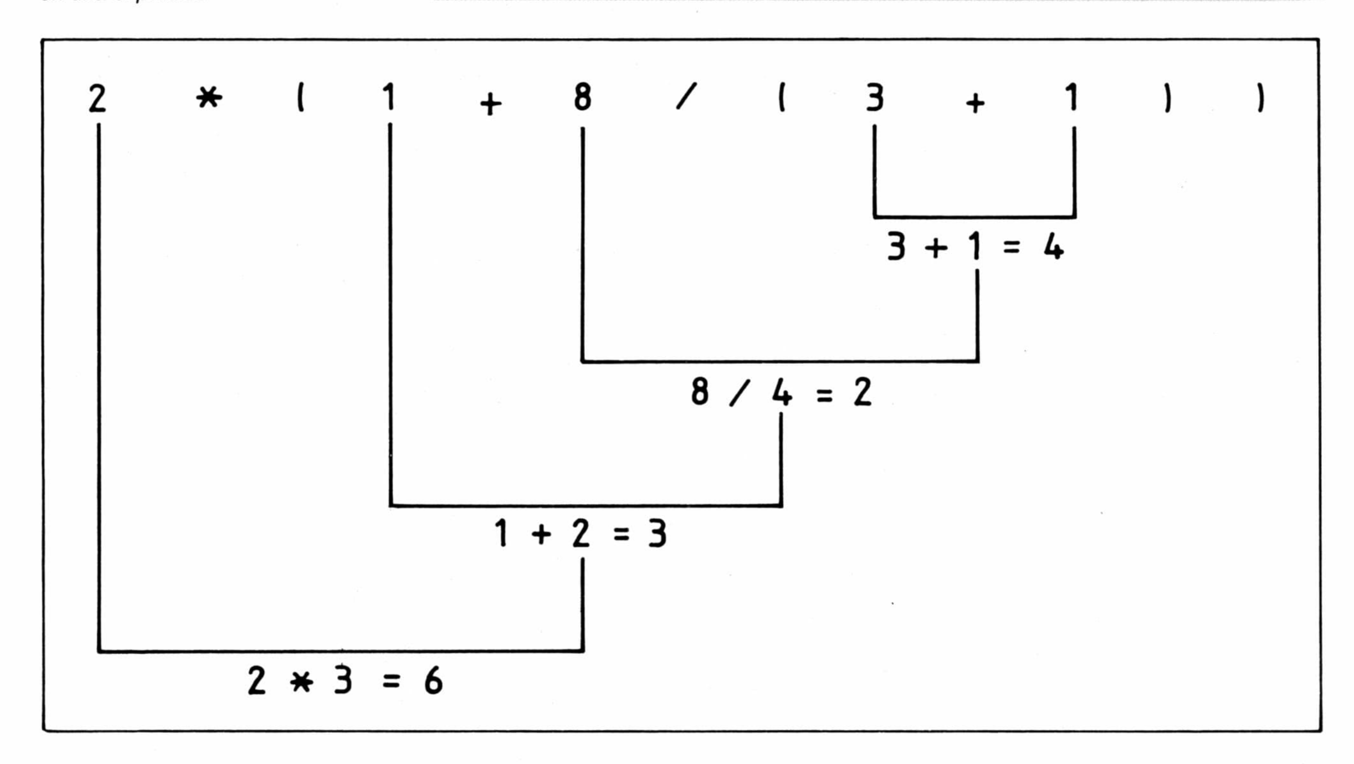


Figura 5 Representación gráfica de la jerarquía de los operadores en una expresión

Siguiendo las reglas anteriores, en esta expresión se efectuará primero la operación 3 elevado a 2 (Nivel 2). El resultado se multiplica por 4 (Nivel 3) y finalmente se le suma 1 (Nivel 4). Quedará más claro si escribimos:

```
PRINT 1+4*(3^2)
```

y más claro aún, si escribimos

```
PRINT 1+(4*(3^2))
```

El resultado es el mismo (37), pero resulta más cómoda la interpretación de su significado.



3.3 REPRESENTACION INTERNA DE LOS DATOS

En algunos apartados anteriores hemos mencionado el sistema binario diciendo que era el sistema utilizado internamente por el ordenador para trabajar. Si bien es cierto que se pueden realizar muchos programas sin necesidad de conocer el sistema binario, no es menos cierto que si se comprende el funcionamiento interno de la máquina es más fácil entender por qué suceden las cosas. Además, esta comprensión nos abrirá la posibilidad de realizar programas para resolver casos especiales e incluso para mejorar la eficacia de los programas corrientes. El primer paso para comprender el funcionamiento del ordenador, es conocer el sistema binario.

3.3.1 Sistemas de numeración

Desde antiguo se han inventado sistemas de numeración para representar todos los números naturales. Algunos, como el romano, asociaban un símbolo a un valor independientemente de la posición relativa que ocupa dicho símbolo. Así, la letra C vale siempre 100 tanto en el número CI como en el MCMLXXXIV. Este sistema era engorroso e inviable para números grandes. Aparecieron entonces los sistemas de numeración basados en la posición relativa. Uno de estos sistemas es el arábigo que es el utilizado actualmente. Como ya conocemos, en este sistema existen 10 símbolos distintos (del 0 al 9) cuyo valor real depende de la posición relativa. Puesto que tiene diez símbolos distintos, se dice que este sistema de numeración tiene la base diez. Este número se eligió porque se utilizaron los dedos de las manos para contar.

En el sistema decimal (base diez) una agrupación de dígitos, como por ejemplo 1792, significa

$$(1\times10^3) + (7\times10^2) + (9\times10^1) + (2\times10^0)$$

$$1000 + 700 + 90 + 2 = 1792$$

Los sistemas de numeración de posición relativa son la base de la Aritmética

Es decir, que cada cifra tiene su valor multiplicado por la potencia de diez correspondiente a su posición.

El sistema basado en el número 10 es muy adecuado para los hombres, pero no lo es tanto para las máquinas ya que es difícil construir circuitos electrónicos que puedan trabajar con este sistema y sean fiables. En efecto, supongamos que tenemos un componente eléctrico de los más sencillos: una bombilla. Por convención consideraremos que si está apagada representa un cero. Si está totalmente encendida un 10. Si sólo está encendida en un 50 % representa un 5, un 40 % un 4 y así sucesivamente. Como es fácil de comprender, una máquina basada en este sistema sería muy poco fiable ya que las graduaciones están muy próximas y cualquier pequeña perturbación (envejecimiento del filamento, fluctuaciones en la corriente, etc.) falsearía los datos.

Para evitar estos problemas se eligió un sistema mucho más sencillo que sólo tuviera dos graduaciones y claramente diferenciables. En el ejemplo de la bombilla, ésta sólo podría estar apagada o encendida. Esto significa utilizar un sistema en base 2, el cual sólo tiene dos símbolos, el 0 y el 1. Por tener la base 2 se denomina sistema binario. Un dígito, en el sistema binario, sólo puede tener dos estados, cero o uno (o SI-NO o CIERTO-FALSO). Este dígito binario representa la unidad básica de información y se denomina BIT (contracción de la palabra inglesa «binary digit»).

3.3.2 Sistema binario

En el sistema decimal cuando sobrepasamos el límite de las cifras (el número 9) empezamos otra vez con el cero y añadimos un 1 a la columna de la izquierda. En el sistema binario se opera de igual forma, pero ahora el límite es el número 1 en lugar del 9. Por tanto, en binario contaremos de la siguiente forma:

Decimal	**Binario**
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
.	.
.	.

Observamos que al sumar una unidad al número binario 11 (3 en decimal) se obtiene el 100 (4 en decimal). Este caso es análogo a sumar una unidad al número 99 en decimal.

En este sistema se sigue cumpliendo lo que hemos dicho sobre la posición relativa de las cifras. Así, el número binario 1001 vale:

$$(1\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0)$$

$$8 + 0 + 0 + 1 = 9$$

Aquí, cada cifra tiene su valor multiplicado por la potencia de 2 correspondiente a su posición.

Conversión de un número binario a decimal

Este procedimiento es el que se utiliza para convertir un número binario en su equivalente decimal. De este modo, si queremos saber cuánto vale el número 1101 en decimal, efectuaremos los siguientes cálculos:

$$(1\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0)$$

$$8 + 4 + 0 + 1 = 13$$

Conversión de un número decimal a binario

Para realizar la operación inversa, es decir, convertir un número decimal en binario se divide el número decimal sucesivamente por la base del sistema binario (2) hasta que el cociente sea cero. Los restos de estas divisiones leídos de izquierda a derecha formando el número binario equivalente. El último resto es el dígito más significativo (a la izquierda del número). Los siguientes restos se colocan consecutivamente hasta llegar al primer resto que es el dígito menos significativo (a la derecha del número). Por ejemplo, convertir el número 227 (decimal) a binario:

```
227 |_2_
02   113 |_2_
 07  013  56 |_2_
  1    1  16  28 |_2_
           0  08  14 |_2_
               0   0   7 |_2_
                       1   3 |_2_
                           1   1 |_2_
                               1   0
```

El número 227 en binario es el 11100011. Es decir, que el último resto será el primer número de la cantidad escrita en binario, el siguiente resto será el segundo número de la cantidad escrita en binario y así sucesivamente. Puede comprobarlo en el ejemplo propuesto.

Como se puede comprobar, en binario los números tienen gran cantidad de cifras, pero en cambio tiene la ventaja de que se adapta perfectamente a su utilización en circuitos electrónicos. Por esta causa, los ordenadores emplean internamente el sistema binario y, por las razones apuntadas anteriormente, se comprende que sean extraordinariamente fiables.

El ordenador trabaja siempre con el sistema binario. Los números que nosotros escribimos son convertidos a binario por el ordenador antes de trabajar con ellos. Posteriormente son reconvertidos al sistema decimal antes de presentarlos en pantalla o impresora.

3.3.3 Agrupaciones de bits

Los múltiplos en el sistema binario son potencias de dos

En informática, los múltiplos no son potencias de 10 (como el km que es 10^3 m) sino de dos. El primer múltiplo es el byte u octeto que contiene $2^3=8$ bits. Con un byte se pueden representar números desde 00000000 (el 0 en decimal) hasta el 11111111 (el 255 en decimal). El siguiente múltiplo es el Kilobyte o KB que vale $2^{10}=1024$ bytes (el prefijo «Kilo» se usa por analogía con el sistema decimal ya que 1024 es muy cercano a 1000). El siguiente múltiplo es el Megabyte o MB que vale 2^{20} (algo más de un millón de bytes, exactamente 1048576 bytes).

Con un byte tenemos 256 combinaciones distintas, de 0 a 255 (esto se puede calcular haciendo $2^8=256$). Para representar números mayores se agrupan varios bytes. El número de bytes empleados determina cuántas cifras exactas tendrán los números utilizados en nuestro ordenador.

Los números fraccionarios se representan en forma exponencial pero en el sistema binario

Hasta aquí hemos hablado de números enteros (sin decimales). Los números con parte fraccionaria también tienen su expresión en binario. Sin embargo, para nosotros no tiene demasiado interés conocer en profundidad cómo se representan dichos números. Simplemente basta con saber que el grupo de bytes que se usan para almacenar un número, se dividen en dos partes. La primera parte, denominada mantisa, contiene las cifras del número junto con su signo más (+) o menos (−), sin especificar la posición de las cifras fraccionarias. La segunda parte, habitualmente formada por un solo byte, se denomina exponente e indica dónde se coloca la coma que separa los decimales.

De hecho, internamente los números fraccionarios tienen una forma análoga a la empleada en la notación exponencial sólo que en binario. Las cifras situadas antes de la letra E constituyen la mantisa y las que están a la derecha, constituyen el exponente.

La memoria de nuestro ordenador está formada por varios KB (varios miles de bytes) y se usa para alcanzar las variables y constantes junto con las instrucciones que forman el programa.

El tamaño total de la memoria y cuántos bytes se usan para memorizar un número son datos que hemos visto en las «Prácticas con el microordenador». Con estos datos calcularemos cuántos números podemos almacenar simultáneamente en la memoria. Por ejemplo, si debemos memorizar 1000 números y cada número gasta 5 bytes, necesitamos un total de $1000 \times 5=5000$ bytes o unos 5 KB de memoria.

3.4 EXPRESIONES TEXTUALES

Los textos están formados por una cadena o ristra de símbolos («string» en inglés). Por esta razón, las variables que contienen textos se denominan a veces variables de cadena.



Al igual que para los datos de tipo numérico, también existen operaciones para las expresiones de tipo textual. Antes de definir las operaciones, conviene que veamos previamente cómo manipula un ordenador los datos textuales o alfanuméricos.

3.4.1 El código ASCII

El conjunto de caracteres admitido por el BASIC incluye las letras mayúsculas (de la A a la Z), minúsculas (de la a a la z), dígitos (del 0 al 9), símbolos de puntuación (como la coma, el punto, etc.) y símbolos especiales (el signo igual (=), el símbolo dolar ($), etc.). En total hay algo más de cien. Como hemos visto en el apartado anterior, el ordenador trabaja exclusivamente en binario. En consecuencia, los caracteres se codifican también en binario.

Cada carácter se almacena en un byte mediante el código ASCII

El conjunto de bits más pequeño que maneja un ordenador de una sola vez es el byte. Con un byte tenemos un total de 256 combinaciones distintas. Como el número de caracteres es inferior a 256, se emplea un byte para almacenar un carácter, asociando una combinación de los 8 bits a cada carácter. Esta asociación se denomina codificación, puesto que atribuimos un código formado por 8 bits a cada carácter. Naturalmente esta asignación es arbitraria. Así, por ejemplo, se podría realizar la siguiente codificación.

10000001 es la letra A

10000010 es la letra B

etc.

El problema está en que si cada ordenador utilizara su propio sistema de codificación, no habría forma de hacer compatibles los datos entre ordenadores distintos e incluso entre ordenadores y periféricos (impresoras, pantallas, etc.). Para evitar este problema se han definido unos códigos estándar, el más extendido de los cuales es el código ASCII. El nombre proviene de American Standard Code for Information Interchange (Código estándar americano para intercambio de información).

En la figura 6 se muestra la tabla del código ASCII, en donde las combinaciones de bits están representadas en su equivalente número decimal. Los 32 símbolos primeros (del 0 al 31) no son imprimibles y se usan como símbolos de control (salto de línea, cambio de página, etc.).

> Cada carácter necesita un byte para ser almacenado. A cada carácter se le asocia una combinación de unos y ceros según un código estándar, que en nuestro caso será el ASCII.

Las variables de tipo textual (cuyo nombre incluye el símbolo dólar ($) al final) ocuparán tantos bytes de la memoria como caracteres tenga el texto a almacenar. Para almacenar la palabra CASA se necesitan 4 bytes.

| C | A | S | A |

La longitud es una propiedad importante de la cadena de caracteres

Una propiedad importante de un dato textual es su longitud, medida en el número de caracteres que lo forman. Esta longitud coincide con el número de bytes necesarios para almacenarlo.

Figura 6 Código ASCII

CODIGOS ASCII DE LOS CARACTERES			
Caracteres de control no imprimibles	**Caracteres imprimibles**		
0 NUL	32	64 @	96
1 SOH	33 !	65 A	97 a
2 STX	34 ″	66 B	98 b
3 ETX	35 #	67 C	99 c
4 EOT	36 $	68 D	100 d
5 ENQ	37 %	69 E	101 e
6 ACK	38 &	70 F	102 f
7 BEL	39 ´	71 G	103 g
8 BS	40 (	72 H	104 h
9 HT	41)	73 I	105 i
10 LF	42 ·	74 J	106 j
11 VT	43 +	75 K	107 k
12 FF	44 ,	76 L	108 l
13 CR	45 -	77 M	109 m
14 SO	46 .	78 N	110 n
15 SI	47 /	79 O	111 o
16 DLE	48 0	80 P	112 p
17 DC1	49 1	81 Q	113 q
18 DC2	50 2	82 R	114 r
19 DC3	51 3	83 S	115 s
20 DC4	52 4	84 T	116 t
21 NAK	53 5	85 U	117 u
22 SYN	54 6	86 V	118 v
23 ETB	55 7	87 W	119 w
24 CAN	56 8	88 X	120 x
25 EM	57 9	89 Y	121 y
26 SUB	58 :	90 Z	122 z
27 ESC	59 ;	91 [	123 {
28 FS	60 <	92 \	124	
29 GS	61 =	93]	125 }
30 RS	62 >	94 ˆ	126 ˜
31 US	63 ?	95 –	127 DEL

El texto de longitud cero es el texto vacío

Así como en los datos numéricos existe el cero, también existe en los datos textuales su equivalente que es el texto de longitud nula (o cero). Para indicar este dato, es decir, un texto, que no contiene ningún símbolo se escriben dos símbolos de comillas ('') seguidos.

```
LET A$=""
```

La variable A$ contiene cero caracteres o, lo que es lo mismo, está vacía.

En general, la longitud máxima de un texto en BASIC está limitada a 255 símbolos.

Se suele considerar que una hoja mecanografiada tiene unos mil símbolos y por tanto, para almacenarla electrónicamente (o magnéticamente) se necesitan unos mil bytes o 1 KB.

3.4.2 El operador de concatenación

Para los datos de tipo textual también se pueden definir operaciones. Las dos operaciones fundamentales son la unión de dos textos para formar uno mayor (llamada concatenación) y la extracción de un segmento del texto.

El operador de concatenación une textos

El operador de concatenación es el signo más (+). En este caso, no tiene el significado de la suma aritmética, sino que significa concatenar (o unir) dos textos. Por ejemplo, si escribimos

```
LET X$="CASA"+"GRANDE"
PRINT X$
```

en pantalla aparece CASAGRANDE. Observamos que no hay espacio en blanco entre la A y la G. Ello es debido a que el operador de concatenación une el último símbolo del primer texto con el primer símbolo del segundo. Si queremos que aparezca el espacio en blanco hay que indicarlo explícitamente. Para ello tenemos tres alternativas: escribir

```
LET X$="CASA "+"GRANDE"
```

o bien

```
LET X$="CASA"+" GRANDE"
```

o bien

```
LET X$="CASA"+" "+"GRANDE"
```

En los tres casos, al efectuar la instrucción PRINT aparecerá el texto CASA GRANDE. En el primer caso se ha dejado el espacio en blanco en el final de CASA, antes de las segundas comillas. En el segundo caso, el espacio en blanco se ha dejado después de las primeras comillas que encierran la palabra GRANDE. En el tercer caso hemos realizado la concatenación de tres textos, uno de los cuales era un espacio en blanco. En este punto hay que recordar dos cosas. La primera es que el espacio en blanco es un símbolo como cualquier otro (el 32 en ASCII) y ocupa un byte como los demás. La segunda es que los textos van siempre entre comillas. Cualquier símbolo, incluyendo el espacio en blanco, que esté fuera de las

comillas no forma parte del texto. En el ejemplo anterior el espacio de separación entre CASA y GRANDE lo hemos situado siempre dentro de las comillas. Por el contrario, si escribimos

```
PRINT "CASA"+      "GRANDE"
```

da como resultado CASAGRANDE ya que los espacios en blanco no forman parte del texto a escribir por estar situados fuera de las comillas.

Para comprobar que el signo más (+) no actúa en este caso de operador aritmético de suma, escribamos las dos instrucciones siguientes:

```
PRINT 1+2
PRINT "1"+"2"
```

La primera escribirá 3 como era de esperar. Pero la segunda escribirá 12, puesto que ha concatenado el uno con el dos. El uno y el dos actúan aquí como constantes textuales puesto que van entre comillas.

Puede parecernos un poco extraña la diferencia entre un 1 y un «1». Recordemos que el BASIC maneja dos tipos de datos totalmente distintos entre sí y que no se pueden mezclar directamente. El primer tipo de dato es el numérico y al cual se aplican las operaciones aritméticas ya conocidas. El segundo tipo es el textual o alfanumérico que utilizamos para guardar palabras, nombres, direcciones, etc.

Para representar los datos numéricos emplearemos los dígitos (de 0 a 9) y el punto decimal (también la letra E para la notación exponencial). Para los datos textuales podemos utilizar cualquier símbolo de los contenidos en el código ASCII, incluidos los dígitos. Pero aquí, los dígitos no tienen significado numérico. Por ejemplo, si consideramos el texto «c/ Mayor 15» está claro que el 15 no tiene sentido como número aislado y a nadie se le ocurriría calcular, por ejemplo, la raíz cuadrada de este dato.

Un texto contiene todos los caracteres que van encerrados entre comillas

Las comillas («) se emplean para indicar al BASIC que se trata de un texto y debe dejarlo intacto. Esto es lo que ha ocurrido al escribir «1»

Un caso que pone de manifiesto claramente la diferencia entre datos numéricos y textuales, es cuando se colocan ceros a la izquierda. Escribamos

```
PRINT 0003
PRINT "0003"
```

En el primer caso se trata de un dato numérico. Por tanto, el BASIC lo analiza y suprime los ceros de la izquierda por no ser significativos. El resultado es el número 3 como vemos en pantalla. Por el contrario, en el segundo caso, el BASIC lo deja intacto por tratarse de un dato textual. En pantalla aparece 0003.

Como ya hemos mencionado anteriormente no se pueden mezclar directamente los dos tipos de datos. Así, sería incorrecto escribir

```
PRINT "c/ Mayor"+15
```

El texto vacío no alarga ningún texto al usarse en la concatenación

Más adelante veremos que hay procedimientos para cambiar el tipo de un dato, y pasarlo de textual a numérico y viceversa.

Si se concatena algo con el texto vacío, representado por dos comillas juntas («»), el texto queda sin variación ya que equivale a sumar cero a un número. Recuerde que el texto vacío no contiene absolutamente ningún carácter. Si entre las dos comillas dejamos espacios en blanco, éste ya no es el texto vacío, sino un texto con varios espacios en blanco y que aparecerán cuando concatenamos con otro texto.

3.4.3 Extracción de segmentos de texto

La operación contraria a la concatenación es la extracción. La forma de expresarla es muy distinta según sea la variante de BASIC que se utilice. Además, puesto que emplearemos funciones para realizar la extracción, reservaremos su estudio para un apartado posterior.



3.5 CASOS PARTICULARES DE LAS EXPRESIONES

Puesto que la precisión de una máquina está limitada a un determinado número de cifras, pueden surgir algunos problemas o comportamientos inesperados en el cálculo de una expresión.

3.5.1 Números muy grandes

Sabemos que el BASIC adopta la notación exponencial cuando un número sobrepasa un cierto tamaño. Por ejemplo, si multiplicamos 1 millón por 1 millón, el resultado se escribe de la forma 1E+12.

La situación de «Overflow» es equivalente a salirnos de límites

Sin embargo, el exponente tampoco puede crecer indefinidamente. Cuando llega a un valor que supera al máximo admitido por el ordenador, se produce un error y aparece el mensaje de «Overflow» u otra indicación equivalente. Esta palabra significa que hemos superado el límite.

Para probarlo, escribamos el siguiente programa

```
NEW
10 LET A=10
20 LET A=A*10
30 PRINT A
40 GOTO 20
```

En la línea 20, el valor de la variable A se multiplica por diez. En la línea 30 se escribe el valor de A a fin de controlar su evolución. En la línea

40 se obliga a repetir el proceso ya que transfiere control a la línea 20. Al ejecutar este programa, al cabo de pocas vueltas, el valor de A supera el máximo y se produce la señal de error.

Es cierto que es difícil que se produzca este error, ya que es poco frecuente la utilización de números tan grandes. Sin embargo, éste no es el único comportamiento inesperado cuando se emplean números grandes.

La suma de números muy grandes y números muy pequeños puede producir comportamientos inesperados

Debido a la limitación en el número de cifras, si se suma un número pequeño a uno grande, éste no sufre ninguna variación.

Ejemplo

```
NEW
10 LET A = 1E+12
20 PRINT A + 1
```

El resultado es 1E+12. Esto es lógico si tenemos en cuenta que intentamos sumar una unidad a la doceava cifra, cuando nuestro ordenador opera con menos cifras.

3.5.2 Números muy pequeños

Cuando se trabaja con números muy pequeños, pueden surgir problemas, especialmente al realizar divisiones. Por ejemplo

```
NEW
10 LET A=1E-35
20 LET B=1E+10
30 PRINT B/A
```

Al realizar la división se produce un error. Este tipo de error es debido a que el resultado de la división es un número demasiado grande. Recordemos también que no es posible efectuar una división si el denominador es cero.

Como en el caso anterior, si se suma un número pequeño a otro mucho mayor que él, éste último no se ve afectado. Por ejemplo

```
NEW
10 LET A = 1E-35
20 PRINT 1+A
```

El resultado dará 1.

De hecho este comportamiento es lógico, ya que el ordenador trabaja con un número determinado de cifras. Así si sumamos 0.0000001 a un 1 en un ordenador de 6 cifras, se obtiene un 1. El resultado que se debería obtener es 1.0000001, pero como sólo puede trabajar con 6 cifras el resultado es 1.00000, el resto de decimales no caben ya que el número supera las seis cifras en total.

3.5.3 Textos demasiado largos

En las expresiones textuales también pueden producirse errores. En la mayoría de variantes del BASIC, la longitud máxima de un dato textual es de 255 caracteres. Si al realizar concatenaciones se supera este límite, se produce o bien un error, o bien el texto queda truncado quedando solamente los 255 símbolos de la izquierda.

RESUMEN

En algunas variantes del BASIC, se puede omitir la palabra LET en las instrucciones de asignación.

Una expresión es un conjunto de operadores y operandos. Las expresiones pueden ser aritméticas o textuales según se apliquen sobre datos de un tipo u otro. Los operadores aritméticos son la suma (+), la resta (−), la multiplicación (*), la división (/) y la potenciación (^). Los operadores son unarios si sólo afectan a un operando y binarios si afectan a dos operandos. Cada operador tiene una jerarquía que determina su prioridad en la evaluación de una expresión compleja. Para cambiar el orden de prioridad se emplean los paréntesis. También se emplean los paréntesis para hacer más legible una expresión compleja aunque no afecten al orden de prioridad.

Los ordenadores trabajan siempre con el sistema binario. El sistema binario es un sistema de numeración basado en el número 2. El dígito binario se denomina Bit y sólo puede tener dos estados, cero o uno. Un conjunto de 8 bits se denomina byte u octeto. Los múltiplos siguientes son el Kilobyte o KB y el Megabyte o MB.

Para almacenar un carácter se emplea un byte. Se asigna una combinación de unos y ceros distinta para cada uno de los caracteres. Esta asignación está definida en el código ASCII.

En las expresiones textuales existe el operador de concatenación (+) que sirve para unir dos textos formando uno de mayor. Este operador utiliza el mismo símbolo que la suma aritmética, pero no debe confundirse con él. Una propiedad importante de los datos textuales es su longitud media en número de caracteres. Esta longitud puede variar entre 0 y 255 símbolos.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

1. Los operadores aritméticos en el BASIC son cinco, la suma, la resta, la multiplicación, la división y la ....

2. Un operador .... es el que necesita dos operandos para que la operación sea correcta.

3. La multiplicación es un ejemplo de operador ....

4. Un operador unario es el que precisa de .... operando para que la operación sea correcta.

5. El operador de cambio de signo es un ejemplo de operador ....

6. Se denomina .... al conjunto de operadores y operandos escritos para realizar un cálculo complejo.

7. La potenciación en BASIC no permite que la .... sea un número negativo.

8. El exponente en la operación de .... puede ser positivo, negativo o incluso cero.

9. El significado de un exponente negativo es .... varias veces la operación de división.

10. El significado de un .... fraccionario, tal como, 1/3, significa sacar una raíz, en nuestro caso la raíz cúbica.

11. Cuando hay que encadenar más de una operación se pueden calcular resultados intermedios mediante la colocación entre .... de la operación que da lugar al resultado intermedio.

12. Los paréntesis se pueden suprimir cuando la jerarquía de los .... contiguos lo permite.

13. El operador suma es de .... nivel jerárquico que el de multiplicación.

14. Los paréntesis que aparecen en una expresión deben estar ....

15. En el interior del ordenador los números se almacenan en el sistema de numeración ....

16. Los bytes son agrupaciones de bits que permiten almacenar un .... mediante el código ASCII.

17. La operación + entre variables textuales significa obtener un texto que consiste en .... los dos textos; el segundo después del primero sin ningún carácter de separación.

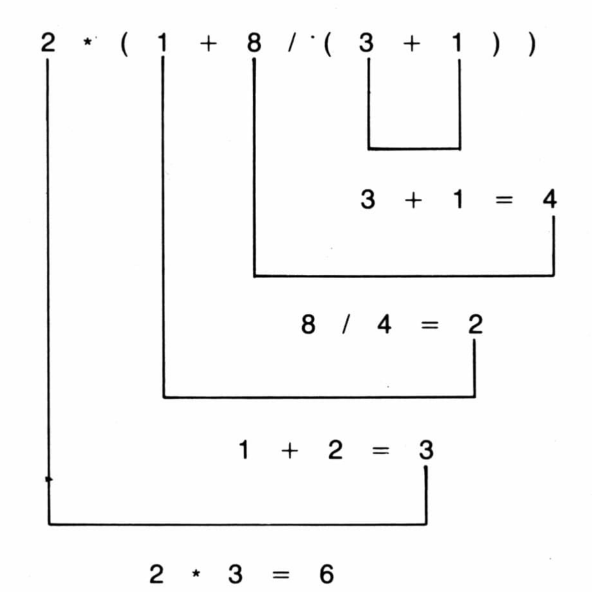
18. Una .... muy importante de los textos es su longitud.

19. El texto de longitud nula es el texto vacío, que se simboliza por dos .... juntas.

20. El texto con varios blancos entre comillas no es el texto ....

Coloque el resultado y el orden de evaluación de las expresiones siguientes tal como se indica en el ejercicio 21.

21. 2*(1+8/(3+1))



22. (5*3−7*4)/(2*2/5−1)

23. 5+2*3^5

24. − (7*3−4) − (5^2)

25. 7−5*3+6^2

Encierre en un círculo la respuesta que corresponda a la alternativa correcta

26. La representación binaria del número decimal 27 es

a) 11111
b) 10101
c) 11011
d) 10001

27. La representación del número decimal 53 es

a) 101010
b) 110110
c) 111111
d) 110101

28. La representación decimal del número binario 11101 es

a) 29
b) 25
c) 14
d) 35

29. La representación decimal del número binario 111111 es

a) 68
b) 63
c) 75
d) 12

30. El resultado de 3^3 es

a) 81
b) 27
c) 9
d) 243

31. El resultado de 4ˆ(−2)

a) 0.25
b) 0.5
c) 0.0625
d) 0.015625

32. El resultado de 81ˆ0.25 es

a) 3
b) 0.3333333....
c) 0.1666666....
d) 9

33. El resultado de 49ˆ(−0.5) es

a) 0.15
b) 0.142857
c) 0.73521
d) 2.78365

34. El resultado de "Juan"+" y " + "Pedro" es

a) JuanyPedro
b) Pedro,Juan
c) Juan y Pedro
d) JuanPedro

35. El resultado de "Juan"+ "" + "Pedro" es

a) JuanyPedro
b) Pedro,Juan
c) Juan y Pedro
d) JuanPedro



3.6 FUNCIONES INTRINSECAS

En el apartado donde hemos estudiado la potenciación, vimos que para efectuar raíces cuadradas se empleaba la función SQR. Podemos considerar a una función como un operador de tipo unario. A este operador se le suministra un dato (denominado argumento) y la función nos devuelve un resultado.

Ejemplo:

```
LET A=SQR(9)
```

Argumento y resultado

Al número 9 (que es el argumento) se le aplica la función SQR y el resultado que es un 3 (que es la raíz cuadrada de 9), se almacena en la variable A. El valor de A se puede consultar con la instrucción PRINT como ya sabemos.

La operación u operaciones realizadas internamente para obtener el resultado dependen de cada función en concreto. En este caso, la operación era extraer la raíz cuadrada (por esto el resultado ha sido 3, ya que es la raíz cuadrada de 9), pero existen muchas más ya que el BASIC dispone de un gran número de funciones.

Al explicar la instrucción PRINT ya había aparecido una de ellas, la función TAB que nos permitía tabular los resultados en una posición concreta. Sin embargo, esta función es un poco especial y sólo se puede utilizar dentro de la instrucción PRINT.

Los argumentos pueden ser expresiones

La forma general para utilizar una función es idéntica a la que hemos visto para la función SQR. En primer lugar se escribe el nombre de la función y a su derecha, entre paréntesis, se coloca el argumento. El argumento, a su vez puede ser una expresión. En este caso, se evalúa en primer lugar la expresión y luego se aplica la función sobre el resultado. Por ejemplo:

```
PRINT SQR(16+9)
```

Aparece un 5 que es la raíz cuadrada de 16+9=25.

Hay funciones que, a partir de un argumento numérico, nos devuelven un resultado también numérico. Existen, asimismo, funciones que de un argumento textual, nos devuelven un resultado textual y finalmente hay funciones que devuelven un resultado de distinto tipo que el argumento suministrado. Por ejemplo, la función que calcula la longitud de una variable textual, se le suministra como argumento un texto y nos devuelve un número. En cambio, la función que transforma una variable numérica en un texto que representa su valor, se le suministra un número y nos devuelve una variable textual. Esta función se denomina STR$.

Recuerde que internamente las variables numéricas están en el sistema binario, para poder manipularlas como texto de cifras decimales es necesario una conversión. Así mediante esta función se pueden obtener textos que contengan la conversión de una variable numérica. Un ejemplo

de este tipo de textos es «Las chocolatinas han costado 123.45 unidades monetarias», que se ha construido juntando mediante la operación más, los textos «Las chocolatinas han costado», el texto que representa en decimal el valor de una variable numérica, obtenido con la función mencionada y el texto «unidades monetarias».

Es decir, escribimos

```
10 LET A = 123.45
20 LET A$ = "Las chocolatinas han costado"
      + STR$(A) + " unidades monetarias"
```

en donde la variable A contiene el número en binario que representa el valor de las chocolatinas.

El tipo de la función (es decir, de su resultado) queda indicado de la misma forma que las variables. Por tanto, las funciones que devuelven un resultado textual llevan el símbolo dólar ($) al final de su nombre. Las que devuelven un resultado numérico como, por ejemplo, la función SQR, no llevan ningún símbolo especial.

Varios argumentos

Finalmente las funciones pueden precisar más de un argumento para su cálculo. Por ejemplo, existe la función RIGHT$ que devuelve un texto construido a partir de otro texto, pero no entero sino sólo los últimos caracteres

Para realizar este cálculo es preciso que se suministre a la función el texto original y el número que indica el número dé caracteres que quiere obtener, es decir, si queremos obtener los 6 últimos caracteres del texto «Las chocolatinas han costado», utilizamos la función RIGHT$ del modo siguiente:

```
LET A$ = RIGHT$("Las chocolatinas
han costado",6)
```

de tal manera que la variable A$ contiene el texto «ostado». (No intente escribir este texto todavía.)

Orden de los argumentos

Por lo tanto, cuando hay que suministrar más de un argumento a una función se colocan en el interior del paréntesis separados por comas. El orden de estos argumentos deben ser estrictamente el que requiere la definición de la función.

En el caso de la función RIGHT$ no es lícito dar primero una variable numérica y a continuación de la coma el texto.

En el caso de varios argumentos es lícito también colocar expresiones en lugar de variables simples. Por ejemplo, es correcto escribir

```
PRINT RIGHT $("Juan"+"Pedro", 5*2-7)
```

(No lo escriba todavía.)

El primer argumento es el texto «JuanPedro», que es el resultado de juntar el texto «Juan» con el texto «Pedro» tal como indica el operador más.

El segundo argumento es 3, que proviene del cálculo de la expresión 5*2−7 (5*2 da 10 y menos 7 da tres).

La función obtendrá los tres últimos caracteres de «JuanPedro», es decir, el resultado sera «dro».

3.7 FUNCIONES NUMERICAS

Estas funciones, a partir de un argumento numérico, retornan un resultado también numérico. Los nombres de las funciones que utilizaremos aquí, son las estándar del BASIC. Las funciones específicas de su microordenador ya las conoce usted por las «Prácticas con el microordenador».

Las funciones numéricas de que dispone el BASIC están ligadas a conceptos fundamentales matemáticos.

En este apartado, se pretende que conozca al menos su existencia pero para muchas aplicaciones de programación son totalmente innecesarias. Adquieren mucho más valor cuando al programación se enfoca la resolución de problemas matemáticos complejos.

Sin embargo, hay algunas que se utilizan frecuentemente en programas de tipo general y que es necesario que conozca bien. Estas funciones son ABS(X), INT(X) y SGN(X).

RESULTADO	**ARGUMENTOS**	
	NUMERICOS	**TEXTUALES**
NUMERICO	ABS(X):Valor absoluto INT(X):Valor entero SGN(X):Signo SQR(X):Raíz cuadrada *ATN(X):Arco tangente *COS(X):Coseno *EXP(X):Exponencial *LOG(X):Logaritmo *SIN(X):Seno *TAN(X):Tangente	LEN(X$):Longitud ASC(X$):Cod. ASCII VAL(X$):Valor
TEXTUAL	CHR$(X):Carácter STR$(X):Valor en texto RPT$(N,X$):Repetición	LEFT$(X$,N):Izquierda RIGHT(X$,N):Derecha MID$(X$,N,M):Centro
	(*)Estas funciones son de índole matemática.	
	FUNCIONES SOLO UTILIZABLES EN LA INSTRUCCION PRINT	
	TAB(X): Tabulador. AT(X, Y): Posicionador de pantalla.	

Figura 7 Clasificación de las funciones intrínsecas

Funciones matemáticas

Una función de cariz matemático pero que es de utilización frecuente es la raíz cuadrada SQR(X).

Finalmente, con el fin de darle información completa sobre el juego de funciones, le mostraremos las funciones típicamente matemáticas, que son: ATN(X), COS(X), EXP(X), LOG(X), SIN(X) y TAN(X).

Si no entiende su significado no se preocupe, sólo los necesitará en problemas de concepción matemática. En este caso, probablemente habrá de estudiar matemáticas antes de poder utilizarlas correctamente, y esto cae fuera de los objetivos de esta Enciclopedia.

A continuación comentaremos el catálogo de las funciones, pero para que tenga una visión global de las funciones a tratar la figura 7 le da el resumen de este catálogo. La figura divide las funciones en cuatro clases, según el tipo de argumentos que se le deben entrar, indicado en la parte superior y el tipo de resultado que da la función, indicado en la parte lateral derecha. En el recuadro inferior se dan las funciones que sólo son utilizables en el PRINT.

Dentro de cada recuadro se coloca el nombre de la función con los argumentos que requiere.

3.7.1 Catálogo de funciones numéricas

ABS(X) Calcula el valor absoluto de X. Si X es positivo, el resultado es el propio argumento X. Si X es un valor negativo, es decir, menor que cero, se le suprime el signo negativo.

```
PRINT ABS(50) escribe  en la pantalla 50
PRINT ABS(-8) escribe en la pantalla 8
```

INT(X) El resultado de esta función es el menor número entero (es decir, sin decimales) más cercano a X. Si X es positivo, equivale a suprimir los decimales, o lo que es lo mismo, obtener la parte entera de X.

```
PRINT INT(3.33) escribe en la pantalla 3
PRINT INT(3.99) escribe en la pantalla 3
```

Como se observa en el ejemplo, no se redondean los decimales sino que simplemente se suprimen.
Para argumentos negativos, el resultado no equivale a suprimir los decimales. Esto se debe a que, para los números negativos, el orden de mayor a menor es inverso que para los números positivos. Así, el número −4 es menor que el −3.

```
PRINT INT(-3.3) escribe en la pantalla -4
```

ya que el menor entero más cercano a −3.3 es el −4.

Se puede describir esta operación del modo siguiente: en primer lugar, se eliminan los decimales; si el número era positivo o nulo, se ha acabado la operación, si el número era negativo se le resta al número resultante de quitarle decimales un 1.

SGN(X) Devuelve el signo de X. Si el argumento es mayor que cero, el resultado es 1. Si X vale cero el resultado es también cero. Si X es negativo (menor que cero) el resultado es −1.

```
PRINT SGN(800) escribe en la pantalla  1
PRINT SGN(0)   escribe en la pantalla  0
PRINT SGN(-38) escribe en la pantalla -1
```

SQR(X) Calcula la raíz cuadrada de X. El argumento X debe ser mayor o igual que cero.

```
PRINT SQR(16) escribe en la pantalla 4
```

ATN(X) Función trigonométrica que calcula el arco tangente. El resultado está en radianes. Las funciones trigonométricas las utilizaremos en el capítulo dedicado a realizar gráficos por ordenador.

COS(X) Función trigonométrica que calcula el coseno de X, donde X es el ángulo medido en radianes.

EXP(X) Función exponencial. El resultado es el número *e* elevado a X (recordemos que e = 2.71828182...). No hay que confundir esta función con la operación de potenciación (expresada con el signo ^). La función EXP eleva siempre el número *e* a un exponente, mientras que la potenciación podía elevar cualquier base a cualquier exponente.

```
PRINT EXP(1) escribe en la pantalla
2.71828182
```

LOG(X) Es el exponente al que hay que elevar el número *e* para que dé X. Por ejemplo, LOG(10) es 2.3025851, que quiere decir que *e* elevado a 2.3025851 es 10.
Esta función se complementa con la anterior. Así, EXP(LOG(X))= X. Es decir, elevar a *e* el logaritmo de 10 es 10. También LOG(e) es 1, pues *e* elevado a 1 es *e*.

```
PRINT LOG(2.718282) escribe en la pantalla 1
```

Para calcular los logaritmos en otra base, dividiremos el logaritmo natural de X por el logaritmo natural de la nueva base.

```
PRINT LOG(2)/LOG(10)
```

obtenemos el valor del logaritmo decimal (en base diez) de 2 que es 0.30103.

SIN(X) Función trigonométrica que calcula el seno del ángulo X, donde X está en radianes.

TAN(X) Función trigonométrica que calcula la tangente del ángulo X, donde X viene dado en radianes.

3.7.2 Ejemplos de utilización

Las funciones se pueden utilizar directamente dentro de las expresiones. Así, por ejemplo, la expresión matemática $Y = 2 \cdot \sqrt{3}$ se escribiría en BASIC

```
LET Y=2*SQR(3)
```

A su vez, los argumentos de las funciones pueden ser expresiones. Por tanto, será válido escribir

```
PRINT INT(10/3)
```

En primer lugar se efectúa la operación entre paréntesis. El resultado de la división es 3.3333. Entonces se calcula la función INT cuyo resultado es 3.

El resultado de una función puede ser un argumento

El argumento puede contener incluso una función con su propio argumento.

```
PRINT SQR(SQR(16))
```

En esta instrucción se evaluará primero SQR(16) cuyo resultado es 4 y seguidamente la raíz cuadrada de 4 que vale 2 que es el resultado final. Observemos que hay el mismo número de paréntesis de apertura que de cierre, es decir que están balanceados. Como ya sabemos este requisito es imprescindible en todas las expresiones que usen paréntesis.

Algunas funciones, como la SQR, no admiten números negativos como argumentos. Para evitar que se produzca un error, utilizaremos la función ABS a fin de suprimir el signo menos (−). Si escribimos el programa

```
NEW
10 INPUT "VALOR";X
20 LET A=SQR(ABS(X))
30 PRINT A
```

sea cual sea el signo de X, no se producirá error.

Redondeo

Muchas veces en la vida práctica se usa el concepto de redondeo, de tal manera que si usted compra, por ejemplo, un cuarto de kilo de manzanas a 17 unidades monetarias el kilogramo, debe pagar 4.25 unidades monetarias.

Si no existen las monedas fraccionarias, usted sólo puede pagar o 4 unidades monetarias o 5 unidades monetarias.

Entonces generalmente se aplica la regla del redondeo, que dice: si la fracción decimal (en nuestro ejemplo es 0.25) sobrepasa el 0.5 (en nuestro ejemplo no lo sobrepasa) al valor sin decimales se le añade 1. Si la fracción decimal no sobrepasa el 0.5 (en nuestro ejemplo no lo sobrepasa) simplemente se eliminan los decimales.

La función INT del BASIC elimina los decimales y se puede utilizar para realizar la operación de redondeo, sin embargo, no lo puede hacer directamente pues debe tener en cuenta el primer caso, en el que hay que sumar uno después de quitar los decimales. Se utiliza de la forma siguiente:

```
INT(X+0.5)
```

En efecto, la manera de utilizar la función anterior da el redondeo, si la variable X tiene una fracción decimal inferior a 0.5 en nuestro ejemplo sólo se tiene 0.25, al sumarle 0.5, nunca sobrepasaremos la unidad siguiente, en nuestro ejemplo da 4.75, que evidentemente no sobrepasa el 5. Al realizar la función INT simplemente quitaremos los decimales y nos quedará 4, que efectivamente es la cantidad que debemos pagar si redondeamos 4.25.

Imagine ahora que el kilogramo de manzana se ha subido a 19 unidades monetarias, para un cuarto de kilo debemos pagar 4.75, que redondeando da 5.

La función que le hemos propuesto actúa del modo siguiente, primero suma 0.5 al número como ahora la fracción decimal sobrepasa el 0.5, el resultado será un número con decimales de una unidad superior, en nuestro ejemplo 4.75 más 0.5 dará 5.25. A continuación la función INT elimina los decimales y nos quedamos con 5 unidades monetarias.

Como práctica escribamos el siguiente programa (tecleando previamente NEW para borrar el anterior):

```
10 INPUT "VALOR";X
20 PRINT INT(X+0.5)
```

Escribiremos RUN y le entraremos un número con decimales para observar cómo realiza la operación de redondear.

Funciones inversas

Finalmente, y a título de curiosidad, algunas funciones matemáticas tienen otras funciones que hacen el cálculo inverso. Por ejemplo, al elevar al cuadrado de la raíz cuadrada nos queda el mismo número.

Pruebe

```
PRINT SQR(2)^2
```

verá que obtiene 2 (quizás 1.9999999 debido al número limitado de cifras que tiene el ordenador, y que depende de cada ordenador).

A estas dos funciones raíz cuadrada y elevar al cuadrado se las denominas funciones inversas.

Las funciones LOG (logaritmo natural) y EXP (exponencial) son la inversa una de otra. Por tanto, la instrucción

```
PRINT LOG(EXP(1))
```

da como resultado un 1, puesto que LOG anula al efecto de EXP. Asimismo, si invertimos el orden de las funciones escribiendo

```
PRINT EXP(LOG(1))
```

el resultado sigue siendo 1. Este efecto también se produce entre las funciones TAN (tangente) y ATN (arco tangente). Por ejemplo

```
PRINT ATN(TAN(1))
```

o bien

```
PRINT TAN(ATN(1))
```

ambas dan como resultado el número 1.



3.8 FUNCIONES NUMERICAS Y ARGUMENTO TEXTUAL

Puesto que el resultado de estas funciones es numérico, sus nombres no llevan el signo dólar ($). El argumento debe ser una constante textual (englobada entre comillas) o una variable de tipo textual (con el signo dólar ($) al final del nombre de la variable).

Estas funciones son importantes en programación y debe hacer un esfuerzo para estudiarlas y conocerlas en profundidad.

Como el resultado es numérico se pueden colocar dentro de cualquier expresión aritmética como resultado intermedio.

3.8.1 Catálogo de funciones

LEN(X$) El resultado de esta función es el número de caracteres que tiene el texto del argumento.

```
PRINT LEN("CASA")
```

El resultado es 4 ya que la palabra CASA tiene cuatro letras.

ASC(X$) El resultado es el código ASCII correspondiente a X$. El nombre de la función (ASC) es una abreviatura de la palabra ASCII.

```
PRINT ASC("A")
```

el resultado es 65. Podemos comprobar que en la tabla de la figura 6, la letra A mayúscula tiene el código 65. Si el texto que se entra como argumento tiene más de 1 símbolo, sólo se evalúa el código del primer símbolo.

```
PRINT ASC("CASA")
```

el resultado es 67 que corresponde a la letra C.

VAL(X$) Esta función permite cambiar el tipo de un dato. Concretamente pasa un argumento de tipo textual a su equivalente numérico. Lógicamente, el argumento debe estar formado por caracteres que tengan significado numérico.

```
PRINT VAL("39")+1
```

El resultado es 40. Si el argumento no tiene significado numérico, por ejemplo

```
PRINT VAL("CASA")
```

no tiene sentido aplicar la función, ya que la palabra CASA no se puede transformar en número. El comportamiento de cada or-

denador frente a un caso como éste, varía desde dar un valor de cero, hasta dar un error.

3.8.2 Ejemplos de utilización

El texto vacío tiene de longitud cero

Recordemos que, dentro de las comillas, todos los caracteres cuentan. Así, si escribimos la instrucción

```
PRINT LEN("    ")
```

donde dentro de las comillas hemos tecleado cuatro espacios vacíos en blanco, el resultado será 4. Si utilizamos el texto vacío (de longitud cero) como argumento, es decir

```
PRINT LEN("")
```

el resultado es cero como corresponde.

El siguiente programa lo utilizaremos para contar el número de letras que contiene un nombre.

```
10 REM Contar letras
20 INPUT "NOMBRE",N$
30 LET L=LEN(N$)
40 PRINT "EL NOMBRE ";N$;" TIENE ";L;
   " LETRAS"
```

La línea 10 es simplemente un comentario sobre lo que realiza el programa. En la línea 20, el programa pregunta por un nombre. En la línea 30 calcula la longitud y la almacena en la variable L. La línea 40 es algo más complicada y constituye un buen ejemplo de cómo construir un texto a base de unir partes fijas con partes variables. Es importante que coloquemos exactamente todos los símbolos de forma precisa tal como están escritos. Nos fijaremos especialmente en la colocación de los espacios en blanco contiguos a los textos. De lo contrario, saldrán todas las palabras pegadas unas con otras.

Para probar el programa, tecleemos RUN y contestemos a la pregunta de la línea 20, el nombre JOSE. El resultado que aparece en pantalla es

EL NOMBRE JOSE TIENE 4 LETRAS

Tal como está construido el programa, se puede emplear también para contar el número de letras (o mejor caracteres) de una frase sin tener que limitarnos a un nombre.

La función ASC la utilizaremos para ver las diferencias de código entre las letras mayúsculas y minúsculas. Entremos el siguiente programa (borrando el anterior).

```
NEW
10 INPUT "LETRA",L$
20 PRINT "EL CODIGO ES ";ASC(L$)
```

En la línea 10, el programa pide que se entre una letra. En la línea 20 nos escribe el código de dicha letra. Teniendo delante la figura 6 lo probaremos para varias letras escritas en minúscula y mayúscula para ver las diferencias de código.



3.9 FUNCIONES TEXTUALES

En estas funciones, tanto el argumento como el resultado son de tipo textual. Se usan para extraer trozos de texto.

Estas funciones son importantes para la programación de la mayoría de problemas; por lo tanto, debe poner la máxima atención para entender su funcionamiento.

La principal dificultad está en que los nombres de las funciones varían mucho de un BASIC a otro, sin embargo, las operaciones que realizan son importantes y muy frecuentes. En primer lugar, debe aprender cuál es el efecto de su aplicación. Después debe preocuparse de traducir el nombre al que posea su BASIC; en el capítulo correspondiente de «Prácticas con el ordenador» encontrará las equivalencias.

Estas funciones tienen más de un argumento. Recuerde que se emplea la coma para separar entre sí los argumentos.

3.9.1 Catálogo de funciones

LEFT$(XS,N) Como su nombre indica («left» en inglés significa izquierda), sirve para extraer un trozo de la izquierda del texto que está como argumento. Se toman los N primeros caracteres de X$. Es importante señalar que X$ no se ve afectado por la aplicación de esta función. Simplemente se realiza una copia de los N caracteres.

```
PRINT LEFT$("CASA",3)
```

El resultado es CAS, que son los tres primeros caracteres de CASA.

RIGHT$(X$,N) Es la función complementaria de LEFT$ («right» en inglés significa derecha). Se toman los N últimos caracteres de X$. Al igual que en el caso anterior, el argumento no se ve afectado.

```
PRINT RIGHT$("CASA",3)
```

El resultado es ASA, que son los tres últimos símbolos de CASA.

MID$(X$,N,M) Sirve para extraer trozos de la parte central de un texto. En este caso hay que indicar dos números, puesto que debemos saber dónde empezamos a contar y cuántos caracteres extraemos, se toman M caracteres de X$ empezando a contar desde N (inclusive).

```
PRINT MID$("CASA",2,2)
```

El resultado es AS ya que contamos dos letras a partir de la segunda (inclusive).
Un caso particular de la función MID$ es la extracción de una sola letra. Para ello se escribe un 1 como último argumento.

```
PRINT MID$("CASA",3,1)
```

En este caso el resultado es una S que es la tercera letra.

3.9.2 Ejemplos de utilización

El programa que se indica a continuación sirve para descompactar una fecha. La fecha está escrita en formato DD-MM-AA, donde DD son las dos cifras del día, MM son las dos cifras del mes y AA son las dos cifras del año, utilizando las funciones LEFT$, MID$ y RIGHT$.

```
10 REM Descompactacion fecha
20 INPUT "FECHA (DD-MM-AA):",F$
30 LET D$=LEFT$(F$,2)
40 LET M$=MID$(F$,4,2)
50 LET A$=RIGHT$(F$,2)
60 PRINT D$;" DEL ";M$;" DE 19";A$
```

Descomposición de un texto en partes

La línea 10 es un comentario identificativo del programa. En la línea 20, el programa nos pide que entremos una fecha escrita en el formato especificado. Observamos que la fecha se almacena en una variable textual (F$) ya que no tiene significado numérico escribir por ejemplo 23-05-86. A partir de este momento se extraen los tres trozos (día, mes y año) que forman la fecha almacenada en F$. En la línea 30 se toman los dos primeros

símbolos que corresponden al día y se almacena en D$. En la línea 40 se toman los caracteres cuarto y quinto que corresponden al mes. Para realizar esta operación empleamos la función MID$ que extrae dos caracteres a partir del cuarto. En la línea 50, extraemos los dos últimos caracteres que corresponden al año.

Finalmente, en la línea 60 se escribe la fecha de forma completa. Si a la pregunta de la línea 20 respondemos por ejemplo 12-10-86, el programa escribirá en pantalla

12 DEL 10 DE 1986

Si probamos este programa para una fecha que tenga el día o el mes inferior a diez, como por ejemplo 07-05-86, el resultado es

07 DEL 05 DE 1986

Sabemos que el cero proviene de que estamos manejando datos textuales. Este es el momento adecuado para utilizar la función VAL que vimos en el apartado anterior. De este modo pasaremos el día y el mes a dato numérico con lo que desaparecerá el cero de la izquierda. Cambiemos la línea 60 por la siguiente

```
PRINT VAL(D$);" DEL ";VAL(M$);" DE 19";A$
```

Probando de nuevo el programa para la fecha 07-05-86 aparece el siguiente resultado

7 DEL 5 DE 1986

El siguiente programa lo utilizaremos para ver las posiciones de las letras de una frase.

```
10 INPUT "FRASE",A$
20 INPUT "POSICION LETRA:",P
30 PRINT MID$(A$,P,1)
```

En la línea 10 entraremos una palabra o una frase. A continuación en la línea 20, el programa nos preguntará la posición de la letra que queremos consultar. Responderemos un número comprendido entre 1 y la longitud total de la frase anterior. En la línea 30, la función MID$ extrae la letra correspondiente a la posición pedida.



3.10 FUNCIONES TEXTUALES Y ARGUMENTO NUMERICO

Estas funciones devuelven un resultado textual a partir de un argumento numérico. Por esta razón llevan un símbolo dólar ($) al final de su nombre.

También son funciones muy útiles en programación y debe poner mucha atención para entender su significado.

3.10.1 Catálogo de funciones

CHR$(X) El nombre proviene de «character» (carácter o símbolo en inglés). Devuelve el símbolo cuyo código en ASCII es X.

```
PRINT CHR$(65)
```

escribe una A mayúscula. Esta función es la inversa de ASC. Por tanto si escribimos

```
PRINT CHR$(ASC("A"))
```

el resultado es una A ya que CHR$ anula el efecto de ASC. Asimismo

```
PRINT ASC(CHR$(65))
```

el resultado es 65 ya que, en este caso, ASC anula el efecto de CHR$.

STR$(X) Sirve para convertir un dato de tipo numérico a textual. El argumento X numérico se convierte en textual. Es la función inversa de VAL.

```
PRINT STR$(1)+"2"
```

El resultado es 12 puesto que el signo más (+) actúa de operador de concatenación.
El nombre de STR$ proviene de «string» que significa hilera o ristra en inglés. La palabra string se usa en inglés como sinónimo de dato textual.

RPT$(N,X$) Esta función no es estándar del BASIC y, por tanto, no todos los ordenadores la incorporan. Consultaremos las «Prácticas con el microordenador» para saber si está disponible en nuestro ordenador. Esta función genera un texto formado por la repetición de N veces el símbolo X$.

```
PRINT RPT$(10,"-")
```

escribe una línea formada por diez guiones (−).

3.10.2 Ejemplos de utilización

Si observamos con detenimiento el código ASCII de la figura 6, veremos que los códigos de las letras minúsculas se obtienen sumando 32 a los códigos de las mayúsculas. Sumando 32 a 65 (letra A mayúscula) se obtiene el 97 (letra a minúscula). Aprovechando este hecho escribamos un programa que transforme una letra mayúscula en minúscula.

```
10 REM Transformacion a minusculas
20 INPUT "LETRA",L$
30 LET C=ASC(L$)
40 LET C=C+32
50 PRINT CHR$(C)
```

La línea 10 es simplemente un comentario. En la línea 20, el programa nos pide que entremos una letra, la cual deberá ser mayúscula. En la línea 30, mediante la función ASC obtenemos el código ASCII de la letra almacenada en L$. A continuación, en la línea 40 sumamos 32 a este número, con lo cual obtenemos el código correspondiente a la letra minúscula. Finalmente, en la línea 50 le hacemos escribir la letra cuyo código hemos calculado.

Si después de teclear RUN, respondemos, por ejemplo, con una T a la pregunta de la línea 20, el programa escribirá una t minúscula.

Centrado de un texto

El programa que se indica a continuación escribe un texto centrado.

```
10 INPUT "TEXTO:",T$
20 LET L=LEN(T$)
30 PRINT RPT$(30,"-")
40 PRINT TAB((30-L)/2);T$
50 PRINT RPT$(30,"-")
```

En la línea 10, entramos el texto que queremos centrar. Este texto, además, estará situado entre dos líneas de guiones.

En la línea 20 evaluamos la longitud del texto entrado mediante la función LEN. Esta longitud la almacenamos en la variable L. La línea 30 escribe una línea de guiones aprovechando la función RPT$. Si nuestro ordenador no dispone de esta función, cambiaremos esta instrucción por

```
PRINT "----------------------------"
```

Es decir, escribiendo entre comillas tantos guiones como le indicábamos a la función RPT$.

Este cambio también se aplica a la línea 50. En la línea 40, calculamos el número de espacios en blanco que hay que dejar para que el texto quede centrado. Si la longitud total es 30 y la longitud del texto es L, nos quedan (30-L) espacios a repartir entre los dos lados del texto. Por tanto, a cada

lado le toca la mitad, o sea (30-L)/2. Este valor es el que colocamos dentro de la función TAB.

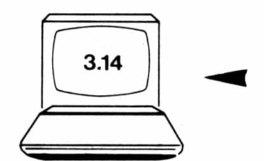
Si entramos el texto CASA, el programa escribe



Si volvemos a probar el programa con otros textos, observamos que siempre quedan centrados.

Tenga en cuenta que cuando el texto posee un número impar de letras el centrado perfecto es imposible en 30 columnas, entonces este programa de centrado deja un espacio menos a la izquierda que a la derecha de la palabra.

La situación se invierte cuando se quiere centrar un texto en un espacio impar de columnas, entonces el centrado perfecto se obtiene para texto con un número de caracteres impar y para los pares hay un carácter más a la derecha que en la izquierda.



3.11 CONTROL DE PANTALLA

Hasta el momento, los resultados que se escribían en pantalla quedaban impresos uno debajo de otro en el orden en que se imprimían. Además, a menudo, los resultados quedaban mezclados cuando, en realidad, nos habría interesado borrar los antiguos antes de escribir los nuevos.

En todos los ordenadores existen dos operaciones básicas. La primera es el borrado de pantalla y la segunda el control de la posición donde se va a escribir. Las instrucciones que realizan estas operaciones varían a veces de un BASIC a otro. En este texto utilizaremos las instrucciones estándar.

3.11.1 Borrado de pantalla

Para borrar la pantalla se utiliza la instrucción CLS. Esta instrucción sólo tiene la palabra clave y carece de complementos.

El nombre proviene de la abreviatura de «Clear Screen» (limpiar la pantalla en inglés). Tecleemos

```
CLS
```

Distinguir entre borrar pantalla y borrar memoria

al apretar la tecla de fin de línea, se borra la pantalla. Es importante señalar que CLS sólo borra lo que está escrito en pantalla, pero no afecta para nada al contenido de la memoria. Por tanto, si tenemos un programa almacenado, éste permanece intacto. Recordemos que la pantalla es un dispositivo periférico del ordenador donde reflejamos datos contenidos en la memoria. El hecho de borrar la pantalla no tiene nada que ver con borrar el contenido

de la memoria. Por el contrario, el comando NEW que ya conocemos sí que borra la memoria.

Como ejemplo de utilización de la instrucción de borrado, escribiremos el siguiente programa

```
10 PRINT "ESTE TEXTO SE BORRARA AL PULSAR"
20 PRINT "LA TECLA DE FIN DE LINEA."
30 INPUT A$
40 CLS
50 PRINT "AHORA SE IMPRIME ESTE TEXTO"
60 PRINT "QUE ES DIFERENTE AL ANTERIOR"
```

Las líneas 10 y 20 escriben un texto. El programa se detiene en la línea 30 preguntando un dato para la variable A$. La variable que se pide no sirve para nada en el programa, pero es una manera de detener el programa en un punto para poder observar lo que ocurre en la pantalla sin prisas.

Al pulsar la tecla de fin de línea, prosigue la ejecución efectuándose la instrucción CLS de la línea 40, con lo cual se borra la pantalla y se imprime ahora el texto de las líneas 50 y 60.

3.11.2 Control de la posición

Existe una función para seleccionar la posición de la pantalla donde se escribirá el siguiente resultado. En primer lugar hemos de imaginarnos que la pantalla está formada por un reticulado construido a base de columnas y filas como se ve en la figura 8. Las dimensiones varían mucho de un ordenador a otro. Entre 30 y 80 para el número de columnas y de 16 a 25 para el de filas o líneas.

La retícula de filas y columnas de la pantalla

Cada uno de los recuadros delimitado por una fila y una columna constituye una posición de la pantalla donde se puede escribir un carácter. Por tanto, para especificar una posición se necesitan dos datos, el número de fila y el número de columna. El funcionamiento es idéntico al de los ejes cartesianos donde para representar un número se indican dos coordenadas.

Para seleccionar una posición en pantalla se usa la función AT («at» en inglés significa «en»). Esta función sólo puede ser usada dentro del PRINT y tiene dos argumentos, puesto que para definir una posición dentro de la pantalla necesitamos dos valores, el primero que indica la columna y el segundo que indica la fila.

```
PRINT AT$(10,10);"HOLA"
```

La palabra HOLA queda escrita en la fila 10 y columna 10 de la pantalla.

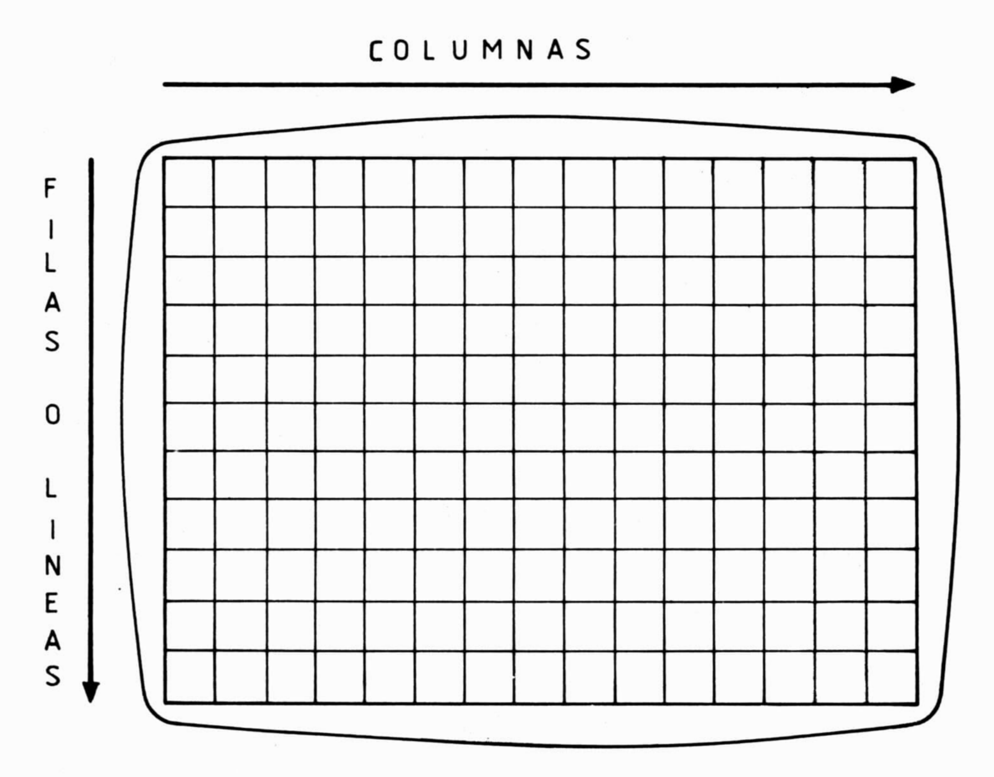


Figura 8 División de la pantalla en filas y columnas

El programa que se da a continuación sirve para observar el efecto del posicionado en la pantalla

```
NEW
10 PRINT AT$(1,20); INPUT "Fila",F
20 PRINT AT$(1,21); INPUT "Columna",C
30 PRINT AT$(C,F);"X";
40 GOTO 10
```

La sentencia 10, entra la fila y la 20 la columna. Es importante señalar que la utilización de la instrucción PRINT previa al INPUT es para que este input se realice en las filas 20 y 21 y no interfieran en la colocación de los distintos puntos en pantalla.

De esta manera se dirige el cursor para la entrada, haciendo un PRINT previo al INPUT, naturalmente este PRINT debe terminar en punto y coma para que el cursor se quede situado en el punto deseado.

La instrucción 30 imprime una X en el punto escogido de la pantalla.

Entre, por ejemplo, para F un 10 y para C un 15, aparecerá una X en la fila 10 y la columna 15. Entre a continuación el 17 y el 19, aparecerá una X en la fila 17 y la columna 19.

Ud. mismo puede entrar otros puntos. Es interesante observar el efecto de entrar números que sobrepasan las dimensiones de la pantalla, en cada ordenador se obtiene un efecto concreto que no es posible generalizar. Debe acabar el programa utilizando el mecanismo de interrupción.

RESUMEN

Una función es un operador que actúa sobre unos argumentos dando un resultado. Las funciones se emplean dentro de las expresiones. La función puede dar un resultado numérico o textual y, a su vez, el argumento puede ser también numérico o textual. Algunas funciones tienen más de un argumento. En este caso se utiliza la coma para separar los argumentos entre sí. Los argumentos pueden ser expresiones que, a su vez, contengan funciones con sus propios argumentos.

Entre las funciones numéricas hay que distinguir de dos tipos, las de uso más general y las que son esencialmente matemáticas.

Las de uso general son las ABS que siempre nos devuelve el número positivo del argumento tanto si es negativo como positivo, la INT que elimina la parte fraccionaria de un número y la SGN que nos devuelve un 1,0 o −1 según el argumento sea positivo, nulo o negativo.

Entre las funciones numéricas de interés matemático están: seno (SIN), coseno (COS), tangente (TAN) y arco tangente (ATN); el logaritmo natural (LOG), la exponencial (EXP), la raíz cuadrada (SQR).

De las funciones matemáticas la SQR que saca la raíz cuadrada es la de más utilidad en la programación en general.

Las funciones textuales se utilizan para extraer segmentos de un texto. Se pueden extraer segmentos de la izquierda (LEFT$), de la derecha (RIGHT$) o del centro (MID$).

Las funciones mixtas que dan un resultado de distinto tipo que el argumento, realizan operaciones auxiliares como calcular la longitud de un texto (LEN), utilizar el código ASCII (ASC y CHR$) o realizar la conversión de tipos (VAL y STR$).

Es un buen momento para volver a repasar la figura 7.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

36. Una función intrínseca es una operación de tipo .... a la que se suministra unos datos y se obtiene un resultado.

37. A los datos que se suministran a una función se les denomina ....

38. Las funciones se escriben con un nombre seguido de paréntesis que encierran los ....

39. Los argumentos pueden ser de tipo numérico o textual, o incluso .... o textuales mezclados según los datos que precisa la función.

40. El .... de una función puede ser de tipo numérico o textual.

41. Los argumentos de una función van separados por ....

42. Cuando se utiliza una función, un argumento puede ser el resultado del cálculo de una .... del mismo tipo (numérico o textual) que el que precisa el argumento.

43. Cuando una función utiliza varios argumentos, la alteración del orden de estos argumentos es un ....

44. Una función puede .... en el cálculo de cualquier expresión.

45. Las funciones se clasifican según el tipo de .... (numérico o textual) y según el tipo de los argumentos (numéricos o textuales).

46. Las funciones numéricas se dividen en funciones generales y funciones de tipo ....

47. Existen funciones inversas, es decir, que una realiza el cálculo .... a la otra.

48. La función .... se denomina de valor absoluto.

49. La función .... se denomina raíz cuadrada.

50. Redondear significa eliminar decimales y si la .... eliminada sobrepasa el 0.5 se añade una unidad.

51. La función .... se denomina longitud de una variable textual.

52. Las funciones textuales sirven para extraer .... de texto de una variable textual.

53. Las funciones CHR$ y ASC son ....

54. Las funciones para convertir un número en texto y un texto en número son la .... y la VAL.

55. Las funciones que sólo utilizan el .... son el AT y el TAB.

Encierre en un círculo las respuestas que corresponden a la alternativa correcta.

56. El resultado de INT (−3*5.2) es:

a) −15

b) −16

c) +2

d) −2

57. El resultado de INT (ABS(SQR(5))) es

a) 2

b) 3

c) −2

d) −3

58. El resultado de SGN (7*4−4−8*3) es

a) 1

b) 0

c) −1

d) Error

59. El resultado de LEFT$(«Hola»+«que tal»,5) es

a) Holaque tal
b) Hola que tal
c) que tal
d) Holaq

60. El resultado de RIGHT$(MID$(«Juan es mi vecino»,3,5),2) es

a) an
b) no
c) es
d) mi

61. El resultado de RIGHT$(LEFT$(«Programa»,5),3)

a) Pro
b) ogr
c) ama
d) ram

62. El resultado de ASC (CHR$(23)) es

a) 24
b) 52
c) 12
d) 23

63. El resultado de CHR$(ASC(«Abanico»)+3) es

a) D
b) F
c) A
d) =

64. El resultado de STR$(7+2*5) es

a) «45»

b) «18»

c) «17»

d) «17.03»

65. El resultado de VAL(«1.3E+2») es

a) 0.13

b) 1.3

c) 13

d) 130

66. El resultado de LEN(RPT$(5,«AB»)) es

a) 5

b) 10

c) 15

d) 20

67. El resultado de RPT$(2,«AB»)+STR$(5*3.2) es

a) AB16AB

b) 16ABAB

c) ABAB16

d) 16

Capítulo 4
• Operadores relacionados

ESQUEMA DE CONTENIDO

4.0 OBJETIVOS

Las decisiones son un elemento importante en todos los programas. Por eso, esta unidad didáctica introduce las bases para que se puedan tomar decisiones.

Usted debe aprender a formular preguntas para que el ordenador las entienda, básicamente, descomponiendo las preguntas complejas en preguntas más simples. El BASIC dispone de unos operadores para contestar estas preguntas simples.

El mecanismo lógico de planteo de preguntas es una de las cuestiones fundamentales en programación y por eso debe hacer el esfuerzo para comprenderlo y aplicarlo. Ya le avisamos que el tema es un poco árido para quien lo estudia por primera vez. Sin embargo, este esfuerzo le dará frutos muy positivos para estar en condiciones de acomodar cualquier problema al ordenador.

Nosotros le enseñaremos a escribir y utilizar expresiones que contengan operadores relacionales en BASIC. No son muy difíciles de entender de modo abstracto, pero casi siempre hay que hacer un pequeño esfuerzo de concreción cuando los aplicamos, para saber si realmente formulamos las preguntas como realmente las queremos formular.

De estos operadores, la igualdad y la desigualdad son los más fáciles de aplicar. Los que corresponden a *mayor, menor, mayor o igual, menor o igual* son algo más difíciles, porque hay que reconocer las propiedades de ordenación de los objetos, que pueden ser números o textos.

Es importante que aprenda a reconocer las propiedades de ordenación de los conceptos que maneja para hacer los programas, y éste es uno de los objetivos importantes del capítulo.

Se estudia con mucho detalle la comparación de textos en BASIC, para que comprenda bien el mecanismo de las comparaciones. No dude en dedicarle el tiempo que se merece.

Finalmente, debe aprender qué mecanismos se siguen para la codificación de las cosas. Esta operación es imprescindible en todos los programas, pues casi todas las asociaciones de los objetos de la vida real, que son los elementos de nuestro programa, se hacen mediante esta operación. Una buena elección del código puede simplificar mucho un programa, y al contrario, una mala elección puede complicarlo enormemente.

4.1 OPERADORES RELACIONALES

Cuando alguien nos pregunta ¿tres es igual a cinco?, la contestación es no, falso. Por el contrario una pregunta: ¿Cuál es el color del vestido?, no tiene una respuesta contundente de sí o no, la respuesta indica un color entre una gama más o menos grande.

No todas las preguntas pueden resolverse de forma simple y contundente, con un sí o un no, con un verdadero o falso.

El ordenador dispone de las operaciones necesarias para responder al tipo de preguntas que sólo tienen dos alternativas como respuesta, el sí o el no, el verdadero o el falso. Las preguntas con sólo dos alternativas como respuesta, se denominan preguntas con respuesta *binaria*.

Preguntas binarias

A primera vista esto puede parecer una limitación importante, pero en realidad no lo es en absoluto. Casi todas las preguntas se pueden cambiar para que la contestación sea un sí o un no. En efecto, podemos reformular la pregunta sobre el color del vestido de la manera siguiente: ¿El color del vestido es azul?

Naturalmente la respuesta se ha reducido en posibilidades de contestación, pues si el vestido no es azul, la contestación no nos informa de qué color es. Para conseguir más información es necesario plantear nuevas preguntas, por ejemplo, ¿El color del vestido es blanco? Si la respuesta es negativa debemos formular una nueva pregunta que pida si el color del vestido es negro, y así sucesivamente hasta encontrar la respuesta afirmativa.

Puede darse el caso de que no acertemos nunca el color, ya que el color real no se encuentra en nuestra gama de colores. La utilización de preguntas complejas puede descomponerse, en la mayoría de los casos, en varias preguntas que sólo permiten un sí o un no como respuesta, es decir, se puede descomponer la pregunta en una *sucesión de preguntas con respuesta binaria.*

Debido a que el ordenador sólo admite las preguntas con respuestas binarias, o lo que es lo mismo con sólo dos posibilidades, el *sí* o el *no,* es una necesidad muy frecuente replantear las preguntas complejas en términos de preguntas de respuesta binaria.

Por ejemplo, la pregunta ¿De qué color es tu vestido?, la hemos descompuesto en la secuencia de preguntas binarias siguientes:

¿El color de tu vestido es azul?

¿El color de tu vestido es blanco?

¿El color de tu vestido es negro?

¿El color de tu vestido es rojo?

¿El color de tu vestido es verde?

La toma de decisiones

La introducción de estas operaciones es necesaria para que el ordenador tome decisiones.

Tomar una decisión consiste en escoger una única alternativa de las varias que tenemos. De una manera muy simplificada, si debemos escoger entre las tres alternativas de ir a Londres, París o Roma, debo escoger una solamente. La respuesta a la pregunta ¿Dónde vamos?, se descompone en las tres preguntas binarias:

¿Voy a Londres?

¿Voy a París?

¿Voy a Roma?

aquella que respondamos con un sí marca la alternativa escogida.

La posibilidad de contestar a más de una con un sí puede considerarse un caso grave de locura. Es evidente que no puedo ir simultáneamente a los tres lugares.

4.2 IGUALDAD Y DESIGUALDAD

Un tipo muy frecuente de preguntas con respuesta binaria son aquellas que piden la igualdad y desigualdad de dos objetos. Consideremos varios ejemplos:

1) ¿Un kilogramo de paja es igual a un kilogramo de hierro?

La respuesta es sí. Se comparan *pesos iguales* de materiales distintos.

2) ¿Un coche azul es distinto a un coche negro?

La respuesta es sí. Se pregunta por la desigualdad de colores. Sin embargo, la pregunta: ¿Un coche azul es distinto a un coche azul?, no es posible responderla con seguridad si no especificamos antes que la única propiedad del coche que nos interesa es el color, prescindiendo de marcas, modelos, tamaños, etc.

Especificar una cualidad precisa del objeto en las comparaciones

Para plantear una pregunta estrictamente binaria, *deberemos especificar de un modo preciso la cualidad de objeto en la que nos fijamos.* Por lo tanto, la pregunta binaria correcta es:

¿El color del coche azul es distinto al color del coche azul?

o, mejor aún,

¿El color del coche A es distinto al color del coche B?

3) ¿Un elefante es igual a una rana?

La respuesta es no. Aunque la respuesta es aparentemente binaria, los objetos no son comparables en la práctica. Surgen los mismos problemas que en el caso anterior. La respuesta a la pregunta ¿Un elefante es igual a un elefante?, sólo es posible si elegimos la cualidad que se quiere comparar: por ejemplo, preguntar por el peso, estatura, longitud de la trompa, etc.

La conclusión que sacamos de la observación de estos tres ejemplos, son las siguientes:

a) La formulación de la pregunta *requiere dos objetos.*

b) Hay que *elegir una propiedad precisa* (color, peso, longitud) que es por la que se va a preguntar.

c) Hay que colocar entre las propiedades de los dos objetos las palabras: *es igual o es distinto.*



4.2.1 La desigualdad en el lenguaje BASIC

El símbolo desigual (<>) es el que utiliza el BASIC para designar el operador de desigualdad, y no el que utilizamos en matemáticas, que es $\neq$. Es el equivalente, a «es distinto a» en las preguntas formuladas anteriormente.

Si queremos comparar números, escribimos:

```
LET A = 3 <> 5
```

Analicemos cuidadosamente esta instrucción. Los números tres y cinco van separados por el símbolo desigual (<>). Esto indica un *operador de tipo binario* (recuerde que un operador binario es el que realiza la operación entre dos objetos). En el apartado anterior hemos concluido que precisábamos de dos objetos para formular la pregunta de desigualdad; concuerda, por lo tanto, con la definición de operador binario.

Finalmente observe que no tiene sentido escribir:

```
LET A = <> 5,
```

que traducido al lenguaje ordinario es equivalente a la pregunta:

¿Es distinto a 5?, ¡qué es lo que es distinto a cinco!, responderíamos coléricos. Por las mismas razones no tiene sentido escribir:

```
LET A = 3 <>
```

La forma de escribir el resto de la instrucción ya la conocemos, significa que la cajita con el nombre A, recibirá lo que se obtiene de la operación 3 es distinto de 5. La variable A es de tipo numérico, es decir, sólo puede contener un número, ¿Cuál es este número?

La respuesta a la pregunta «algo es distinto de algo», sólo tiene dos posibilidades, un sí o un no.

El SI y el NO se representan numéricamente

Recordemos que la definición de bit, es un objeto que contiene información con sólo dos posibilidades de valores 0 y 1; la semejanza con el NO y el SI es muy clara. Efectivamente, si la pregunta se responde con un SI, colocaríamos un bit de la memoria en 1, si se responde con un NO colocaríamos un 0. La memoria en forma de bits es muy adecuada para responder a estas preguntas. Ya sabemos que está dividida, por necesidad en las otras operaciones, en octetos (bytes). Por otra parte, el BASIC sólo trabaja con números y textos. La solución que se adopta es representar el SI mediante un número y el NO mediante otro número. Aunque la variable A es una cajita que puede contener muchos más números, si alberga el resultado de una comparación de desigualdad, sólo contendrá uno de los dos números (el 0 o el 1), de todos los posibles.

En todos los dialectos del BASIC, la respuesta NO se simboliza con un cero. Desafortunadamente en la respuesta SI no existe unidad de criterios. En la mayoría se utilizaría el −1 como número que representa el SI, mientras que en algunos pocos se utiliza el 1.

Si consideramos ahora, la instrucción:

```
LET A = "GARCIA" <> "GARCIA"
```

se pueden hacer las mismas reflexiones que en el caso de los números.

Se precisan dos textos porque el operador es binario. Tampoco tiene ningún sentido la instrucción:

```
LET A = <> "GARCIA"
```

o la instrucción:

```
LET A = "GARCIA" <>
```

El resultado es una variable numérica que representa mediante un cero, la contestación NO y mediante un 1 o −1 la contestación SI.

Por lo tanto, hemos introducido un nuevo operador binario, que es el operador de desigualdad que se simboliza en el BASIC por el símbolo desigual (<>). Este operador se coloca entre dos constantes o dos variables del mismo tipo, es decir, ambas numéricas o ambas textuales. El resultado, o es el número 0, para indicar una respuesta negativa, o es el número −1 o 1 (según que dialecto del BASIC) para indicar la respuesta afirmativa.

Para no ir repitiendo constantemente que la contestación SI se representa por un 1 o un −1 según el ordenador, utilizaremos para este tipo de resultados los nombres de SI y NO. Tenga en cuenta que si un dialecto del BASIC decide representar con un −1 al SI, lo hará siempre de esta manera, nunca a una contestación SI el ordenador le dará un valor distinto a −1. Lo mismo puede decirse en el caso de elegir como representación del SI el 1.

Volvamos a los ejemplos anteriores, consideremos primero la comparación entre números,

```
LET A = 3 <> 5.
```

tecleemos después la instrucción:

```
PRINT A
```

el resultado que aparecerá en la pantalla es un 1 (o −1) que representa el SI; efectivamente el número 3 es distinto del número 5.

En el ejemplo de las constantes textuales, la instrucción:

```
LET A= "GARCIA"<>"GARCIA"
```

pedimos entonces la operación:

```
PRINT A
```

el valor que aparece en la pantalla es 0, pues la respuesta a ¿«GARCIA» es distinto a «GARCIA»? es NO. Observemos que la afirmación de que el texto «GARCIA» *es igual* a «GARCIA», es lo mismo que afirmar que «GARCIA» *no es distinto* a «GARCIA».

Veamos algunos ejemplos más:

```
PRINT 67<>67                 Aparecera en pantalla   NO
PRINT "GUAPO"<>"FEO"         Aparecera en pantalla   SI
PRINT -3<>3                  Aparecera en pantalla   SI
PRINT "AVE"<>"AVE      "     Aparecera en pantalla   SI
```

hay que observar que, efectivamente, los números negativos y positivos son distintos, es el caso del tercer ejemplo. Por otra parte, si a un texto le añadimos blancos al final, como es el caso del último ejemplo, el resultado es SI porque el segundo texto es distinto al primero; precisamente porque se han añadido blancos al final.

La comparación debe hacerse entre variables del mismo tipo

Finalmente conviene aclarar que no es posible utilizar el operador desigualdad para comparar dos constantes o variables de tipo distinto, o sea, una numérica y una textual. La instrucción:

```
LET A = L <> "GARCIA"
```

no tiene ningún sentido, pues L es una constante numérica y «GARCIA» en una constante textual. La comparación es semejante a la de un elefante y una rana, que de hecho no son comparables, es cierto que son distintos pero ciertamente, en ninguna circunstancia serán iguales, *la respuesta no es binaria, no tenemos alternativa.*

El lenguaje BASIC da un mensaje de error cuando se intenta una comparación de este tipo. En este sentido, es mucho más estricto que en el lenguaje ordinario, y diríamos que actúa así por principio, independientemente de las posibilidades de significado al comparar cosas parecidas. Por ejemplo, la instrucción,

```
LET A = 3 <>"3"
```

puede parecernos correcta. El BASIC reacciona con la misma severidad que el caso de L <> «GARCIA», porque sabe muy bien que una cantidad numérica y otra textual no son comparables en absoluto.

El planteo correcto de una instrucción de comparación de dos variables de distinto tipo tiene dos alternativas:

Primera: Utilizar la función VAL, descrita en la unidad didáctica 3, que transforma una variable textual en una numérica con el valor que representa el texto. Así, si escribimos:

```
LET A = 3 <> VAL("3")
```

La utilización de la función Val para las comparaciones

la instrucción no es errónea, pues la variable textual se ha convertido en la numérica correspondiente a su valor. Recuerde, no obstante, que esto no es siempre posible; así la ejecución de la instrucción VAL(«GARCIA») nos dará error pues el texto «GARCIA» no tiene ningún significado numérico. Este efecto es muy importante en un programa, por ejemplo:

```
NEW
10 INPUT A$:REM Se introduce una cadena.
20 INPUT B : REM Se introduce un numero
30 PRINT B<>VAL(A$):REM Se comparan mediante
   la funcion VAL
40 GOTO 10
```

El programa funcionará correctamente mientras entremos textos, tales como «3», «67», «456», pero en cambio la sentencia 30 nos dará error si introducimos «GARCIA», «PEPE». ¡Hágalo! Tenga en cuenta que en la variable A$ nos pide un texto y que, aunque le demos un número, lo toma como texto. Sin embargo, en la variable B nos pedirá siempre un número.

Segunda: Utilizar la función STR$ para transformar la variable numérica a textual, así si escribimos

```
LET A = STR$(3)<>"3"
```

La utilización de la función STR$ para las comparaciones

la comparación adquiere sentido porque se comparan dos variables textuales. En el ejemplo del programa, ahora no tendremos problemas pues siempre una variable numérica se puede transformar en textual, la nueva versión sería:

```
NEW
10 INPUT A$
20 INPUT B
30 PRINT A$<>STR$(B)
40 GOTO 10
```

que no dará error en la instrucción 30 aunque entremos «GARCIA» en la variable A$.

Otras dificultades pueden surgir en casos como el siguiente. Entremos en la instrucción número 10, «3.0» y en la instrucción número 20, 3; el resultado de la comparación es SI pues STR$ es «3» que efectivamente es distinto de «3.0», si se contemplan como textos, aunque en valor numérico significan lo mismo.

Comparando estas alternativas con el ejemplo de la rana y el elefante, la utilización de las funciones VAL y STR es equivalente a especificar una propiedad de la rana y de los elefantes que sean comparables, por ejemplo, el peso, el color, la altura. De modo que la pregunta de desigualdad entre una rana y un elefante se transforma en correcta si decimos:

El peso de la rana es distinto del peso del elefante.

La altura de la rana es distinta de la altura del elefante.

El color de la rana es distinto al color del elefante.

4.2.2 La igualdad en lenguaje BASIC

El símbolo igual (=) es el que utiliza el BASIC para designar el operador de igualdad. Es equivalente a «es igual a» en las preguntas del lenguaje ordinario.

Del mismo modo que el operador desigualdad (<>), el operador igualdad es un operador relacional, ya que pone de manifiesto una relación entre dos objetos, la relación de identidad.

El operador igualdad (=) es un operador binario, que compara dos variables o constantes del mismo tipo y que da como resultado una variable numérica de valor 0, si la contestación es NO y 1 o −1, según el ordenador, si la contestación es SI.

La igualdad en el operador contrario a la desigualdad

Comparte todas las propiedades del operador desigualdad que ya hemos mencionado, excepto que *el resultado es justamente el contrario del que se obtiene con el operador desigualdad,* tomando los ejemplos que se han discutido en el operador desigualdad,

```
PRINT 67=67                Aparecera en pantalla un SI
PRINT "GUAPO" = "FEO"      Aparecera en pantalla un NO
PRINT -3=3                 Aparacera en pantalla un NO
PRINT "AVE"="AVE      "    Aparecera en pantalla un NO
```

se observa que las contestaciones que eran en el operador desigualdad un SI, se han transformado en un NO en el operador igualdad y todas las que en la desigualdad eran un NO, se han transformado en un SI en la igualdad.

Una dificultad que se encuentra en el operador igualdad es que es un símbolo que ya se ha utilizado en otra instrucción en BASIC. Recordemos la instrucción:

```
LET A= 3
```

El igual como símbolo de comparación y asignación

en este caso, como se ha explicado en la lección 1, el igual (=) no se ha denominado operador igualdad sino asignación de un 3 a la casilla A.

Parece que entramos en un callejón sin salida, el símbolo igual (=) puede significar dos cosas distintas en un mismo lenguaje; a un símbolo de este tipo se le denomina *ambiguo,* pues puede significar dos cosas distintas, según donde lo empleemos. Esto es común en el lenguaje corriente. En efecto, consideremos estas dos frases:

Mi padre me ha *dicho* que llegará tarde.

Quien a buen árbol se arrima buena sombra le cobija, es un *dicho* castellano.

En cada una de ellas aparece la palabra *dicho,* pero su significado es muy distinto en cada una de las frases. En la primera *dicho* significa comunicado, mencionado. En la segunda significa proverbio, refrán. Incluso podemos imaginar una frase como

Luis me ha *dicho* un *dicho*

que, aunque suena mal a nuestro oído, pues se parece un poquito a un trabalenguas, tiene un significado correcto y preciso. Averiguamos el significado de *dicho* por su colocación en la frase, y por su significado gramatical. Como ya sabe en la primera frase se utiliza dicho como verbo, mientras que en la segunda frase se utiliza como nombre.

El lenguaje BASIC procede de la misma manera, en la instrucción

```
LET A= 3=5
```

el primer igual (=), por ser el primero precisamente, significa símbolo de asignación, mientras en el segundo, es decir, por no ser el primero significa operador de igualdad. El resultado que se coloca en la variable A es de un NO, pues efectivamente el 3 no es igual al 5.

La posición del símbolo igual (=) en una instrucción LET determina su significado. La regla es la siguiente:

Si el símbolo igual (=) se encuentra en primer lugar en una instrucción LET, significa *asignación a.*

Los demás símbolos igual (=) que aparezcan en una instrucción LET significan operador relacional de igualdad, es decir, *es igual a.*

Por otra parte, debemos observar que en una instrucción PRINT, el símbolo igual (=) es siempre un operador relacional ya que la instrucción PRINT no puede contener un operador de *asignación a.*

Así, en la instrucción siguiente (no la escriba aún, ya que formará parte de un programa del ejemplo que veremos a continuación):

```
PRINT A=B
```

el ordenador escribiría un NO o un SI según el contenido de la variable A fuera distinto o igual al contenido de la variable B.

El programa siguiente nos ayudará a estudiar los diferentes resultados de la comparación de igualdad y desigualdad.

```
NEW
10 INPUT A :REM se introduce un primer numero.
20 INPUT B :REM se introduce un segundo numero.
30 PRINT A=B, A<>B : REM se imprimen los resultados de = y <>.
40 GOTO 10 : REM se vuelve a empezar otro ejemplo.
```

Introduzcamos los datos siguientes:

Datos a introducir A	B	EN PANTALLA APARECE Columna 1	Columna 2
3	3	1	0
5	7	0	1
–4	4	0	1
–0	0	1	0

Observemos que los resultados de la comparación de igualdad y desigualdad son contrarios. También hay que resaltar que la comparación del −0 y el 0 dan igualdad, pues el único número cuyo valor positivo es igual a su valor negativo es el cero.

Ahora pulse la tecla de interrupción para salir del programa que hemos utilizado para analizar el comportamiento de los operadores de igualdad y desigualdad.

RESUMEN

Las preguntas binarias, es decir, las que tienen como respuesta únicamente dos alternativas son las únicas que el ordenador sabe responder.

Todo problema de toma de decisión requiere descomponer una pregunta compleja en una sucesión de preguntas binarias.

Tomar una decisión consiste en elegir un camino de las varias alternativas que pueden existir.

La igualdad y la desigualdad son las preguntas binarias más fáciles sobre dos objetivos. La igualdad responde a la pregunta ¿un objeto es igual a otro?, y la desigualdad responde a la pregunta ¿un objeto es distinto a otro?

En estas preguntas son necesarios dos objetos y además debe especificar la cualidad precisa del objeto que se compare. Preguntas aparentemente binarias, tales como: ¿Un avión es igual a un barco?, no lo son pues no se comparan cualidades similares de los objetos, por ejemplo ¿la longitud de un avión es igual a la longitud de un barco?

El BASIC utiliza el símbolo <> para la desigualdad y el símbolo = para la igualdad. Estos símbolos son operadores de tipo binario, es decir, se deben colocar entre dos variables o constantes del mismo tipo. No son aplicables entre una variable textual y una numérica o viceversa. El resultado de este operador es un número que sólo tiene un valor que llamamos NO y que vale cero en todos los BASIC, o un valor que llamamos SI que en algunos BASIC vale 1 y en otros −1.

En cuanto al operador de igualdad hay que distinguirlo del operador de asignación. Concretamente en la instrucción LET, siempre el primer igual significa asignación, los restantes son operadores relacionales. El símbolo = es ambiguo y debe apreciarse su significado exacto por la posición en que aparece.

Los operadores de igualdad y desigualdad son contrarios, de tal manera que si el resultado para uno es SI, para el otro es NO y viceversa.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

1. Las preguntas .... sólo tienen como respuesta dos alternativas.

2. Tomar una decisión consiste en escoger una .... de las varias posibles.

3. Las preguntas complejas hay que descomponerlas en una .... de preguntas binarias para poder utilizarlas en un ordenador.

4. Los dos primeros operadores relacionales son los que se refieren a los conceptos de .... y desigualdad.

5. Para que las preguntas de igualdad y desigualdad entre dos cosas tengan sentido, deben referirse a una .... específica del objeto.

6. El BASIC utiliza el símbolo .... para el operador de igualdad y el símbolo .... para la desigualdad.

7. La igualdad o desigualdad debe plantearse entre dos variables del mismo ...., es decir, entre variables numéricas o variables textuales, pero no mezcladas.

8. El símbolo = es ambiguo en BASIC, ya que se utiliza para la igualdad y para la asignación. Hay que saber a cuál se refiere según la .... y la .... que ocupa.

9. Los resultados de los operadores igualdad o desigualdad son siempre .... y sólo tienen dos valores, el NO y el SI.

10. En algunos ordenadores el SI se representa mediante un .... y en otros mediante un ....

Encierre en un círculo la respuesta que corresponda a la alternativa correcta.

11. La igualdad y la desigualdad son operaciones:

   a) binarias
   b) unarias
   c) aritméticas
   d) lógicas

12. En BASIC los operadores que responden a preguntas binarias se les denomina:

   a) aritméticos
   b) booleanos
   c) textuales
   d) relacionales

13. En las preguntas siguientes hay una que no es binaria, ¿sabe reconocerla?

   a) ¿De qué color es el mar?
   b) ¿El color del mar es azul?
   c) ¿El color del mar es igual al color del cielo?
   d) ¿El color del mar es distinto al color del cielo?

14. El operador de desigualdad se simboliza en BASIC:

    a) +

    b) <>

    c) NONEQUAL

    d) /<

15. En las preguntas siguientes hay una que no es binaria, ¿sabría reconocerla?

    a) ¿El peso de Juan es igual al peso de María?

    b) ¿El cabello de Juan es distinto al cabello de María?

    c) ¿El color de los ojos de Juan es igual al color de los ojos de María?

    d) ¿Juan y María son igualmente altos?

En las preguntas binarias siguientes, rodee con un círculo la S, si la respuesta es cierta, y la N si es falsa.

16. ¿El número 3 es igual al número 5? S N

17. ¿El número 3 es distinto al número 5? S N

18. ¿El texto «GARCIA» es distinto al texto «GARCIA»? S N

19. ¿El resultado de 3×4 es distinto al resultado de 2×6? S N

20. ¿El resultado de «JUAN»+«PEDRO» es igual al resultado de «PEDRO»+«JUAN»? S N

21. ¿El código numérico de NO es distinto de cero? S N

22. ¿El resultado de VAL («5*4») es igual a 20? S N

23. ¿El peso de 1 kg de plomo es distinto al peso de 1 kg de acero? S N

24. ¿La situación de Washington es igual a la situación de Moscú? S N

25. ¿El número del ejercicio que estoy haciendo es igual a 25? S N

4.3 COMPARACIONES ENTRE CONJUNTOS ORDENADOS

En el apartado anterior hemos visto las comparaciones de igualdad y desigualdad. Vamos a estudiar en este apartado otro tipo de comparaciones, las de mayor que, mayor e igual que, menor que y la de menor e igual que.

Recordemos que en lógica estricta la pregunta ¿es una pera mayor que una manzana?, no tiene respuesta pues se comparan cosas de naturaleza distinta. Hemos visto que estas preguntas se pueden formular correctamente si establecemos una cualidad determinada para usar en los términos de comparación. Por ejemplo, si escogemos como cualidad el peso, es correcto hablar de si el peso de una pera es mayor o menor que el peso de una manzana.

El mecanismo ha consistido en reducir el criterio de la cualidad peso a un número que nos ha suministrado una balanza, en realidad, por lo tanto comparamos números. Para que la comparación sea correcta estos números deben significar el peso en las mismas unidades. Si utilizamos kilogramos, ambos pesos deben ir en kilogramos; si utilizamos gramos, los pesos de la pera y la manzana deben ir en gramos. La comparación será incorrecta si medimos el peso de la manzana en gramos y el peso de la pera en kilogramos.

De hecho todas las comparaciones que se utilizan en programación se reducen a comparar números, que representan alguna propiedad de algún objeto, tal como el peso, en el ejemplo mencionado.

Los numeros son el conjunto ordenado más representativo

Un número siempre se puede comparar a otro porque pertenecen a un conjunto de cosas que están ordenadas. La manera más sencilla de comprender un conjunto de cosas ordenadas es precisamente reducir la propiedad de orden a un número. Si tomamos dos números cualesquiera siempre podemos decir si un número es mayor, igual o menor que otro número. Esta es la regla de oro para saber si un conjunto de cosas es ordenado: si tomamos dos cosas de este conjunto siempre podemos decir si una es mayor, igual o menor que la otra.

De manera muy general, todas las propiedades de los objetos que se puedan representar mediante un número, presentan las características de conjuntos ordenados. Hay infinidad de ejemplos en los que la propiedad puede reducirse a un número, por ejemplo, peso, longitud, altura, profundidad, temperatura, superficie, volumen, velocidad, etc.

A los objetos se les asocian propiedades que provienen de una medida

El proceso de reducir una propiedad a un valor numérico que presenta características de ordenación se denomina *medida.*

No siempre es posible realizar una medida. Pensemos, por ejemplo, en propiedades tales como el color, donde no es posible asignar un número a cada color; el sabor, donde no es posible asignar a los manjares una medida de sabor; el dolor, donde no es posible saber si una muela nos duele más que la del compañero (sinceramente, siempre creemos que la nuestra duele más que la de otro). Todos estos ejemplos presentan propiedades de naturaleza psicológica, es decir, no definible en números o lo que es lo mismo no está establecido un proceso de medida.

No siempre éste es el caso, consideremos la posición de los barcos que se obtiene de lectura de los números de unos aparatos marinos llamados brújula y sextante. La posición de los barcos se mide por la latitud y la longitud, por ejemplo 30 grados latitud Norte y 60 grados longitud Este. Aunque esta posición se reduce a números no es correcto establecer si la posición de un barco es mayor o menor que la de otro barco. En todo caso, sólo podemos preguntar si están en la misma posición o en posiciones diferentes, es decir, sólo se pueden establecer comparaciones de igualdad y desigualdad. La causa de este comportamiento es que de la medida se obtienen dos números, no uno sólo, en el ejemplo anterior 30 y 60.

Por lo tanto, para poder establecer comparaciones es necesario poder medir algo en un solo número y sólo uno.

4.3.1 La comparación de números en BASIC

En BASIC se plantean las preguntas de comparación entre números con los cuatro operadores siguientes:

Nombre	**Símbolo**
Mayor que	>
Mayor o igual que	>=
Menor que	<
Menor o igual que	<=

Estos operadores son binarios, requieren dos números entre los cuales los vamos a aplicar, igual que los operadores de igualdad y desigualdad. El resultado de la operación es una respuesta binaria, es decir un SI o un NO, que se representan por el número −1 o 1 —según el ordenador— para el SI, y con el número 0 para el NO. Sin embargo, los operadores de comparación se utilizarán entre conjuntos ordenados, tal como veremos más adelante.

Para practicar un poco con estos operadores construyamos el programa siguiente, que nos dará la respuesta SI o NO para la comparación de dos números, según el operador de que se trate.

```
NEW
10 INPUT A,B : REM Entramos dos valores numericos.
20 PRINT A=B, A<>B
30 PRINT A>=B, A>B
40 PRINT A<=B, A<B
50 GOTO 10 : REM Enviamos a pedir mas numeros.
```

En las sentencias 20 a 40, calculamos los resultados de todos los operadores relacionales, incluidos la igualdad y la desigualdad.

Tecleemos el RUN, e introduzcamos los valores 3 y 5, es decir, la variable A contiene un 3 y la variable B contiene un 5, en pantalla aparecerán los resultados siguientes:

```
0    1
0    0
1    1
```

El primer NO, es decir el cero, indica que la respuesta a la pregunta: ¿3 es igual a 5? A=B, es NO; el 1 de la misma línea indica que la respuesta es SI a la pregunta ¿el tres es distinto de 5?, A<>B.

El cero de la segunda línea indica que la respuesta es NO a la pregunta: ¿El número 3 es mayor o igual que 5?, A>=B; el segundo cero de la segunda línea indica que la respuesta a la pregunta: ¿El 3 es mayor que 5?, A>B, es NO.

Los 1 de la tercera línea indican que la respuesta es SI a las preguntas: ¿El 3 es menor o igual que 5?, A<=B, y a la pregunta: ¿el 3 es menor que el 5?, A<B.

Los operadores relacionales utilizan más de un símbolo

Es importante darse cuenta de que los operadores que hemos introducido utilizan dos símbolos para expresarse; de hecho ya lo habíamos visto en la desigualdad (<>), esto no nos debe extrañar pues también, por ejemplo, las instrucciones PRINT e INPUT contienen más de una letra y además comparten las letras I y P. De la misma manera los símbolos <=, >=, <> y = son operadores distintos aunque entre ellos tengan símbolos iguales.

Esta construcción de símbolos no debe confundirnos al interpretar las instrucciones. Si consideramos la instrucción:

```
LET A<=3
```

el igual que aparece en la instrucción no se refiere a un símbolo de asignación pues va precedido por el símbolo <, luego la instrucción es incorrecta porque una instrucción LET siempre precisa de un símbolo de asignación.

Continuemos haciendo pruebas con el programa. Tecleemos ahora el número −6, que se colocará en la variable A y el número 2, que se colocará

en la variable B; al apretar la tecla de fin de línea en la entrada del número 2, aparecerá en pantalla:

```
0      1
0      0
1      1
```

Los resultados se interpretan correctamente si se sabe que el número −6 es menor que el número 2, los operadores igual (=), mayor o igual que (>=) y mayor que (>) dan NO y los operadores distinto (<>), menor o igual que (<=) y menor que (<) dan SI. Se desprende de este ejemplo que los números negativos son menores que los números positivos.

Tecleemos ahora el número −4 y el número −10, a diferencia del caso anterior ahora los dos números son negativos, el resultado del programa es el siguiente:

```
0      1
1      1
0      0
```

Los resultados confirman que el número −4 es mayor que el número −10; en efecto, se obtiene un NO para los operadores =, >= y > y un SI para los operadores <>, <= y <. La regla que se desprende es que en la comparación de números negativos, el número mayor sin el signo menos es el número menor con el signo menos.

Para facilitar la memorización de estas reglas recuérdese el funcionamiento de un termómetro, la tempertura de −10 es más fría que la temperatura de −4.

Los números negativos son menores que los positivos

La línea siguiente da idea de esta propiedad de los números negativos.

−500 ...−98 ...−2,−1,0,1,2,... 98... 500
Menor ————————————→ Mayor

los números más pequeños están situados a la izquierda del diagrama, los mayores a la derecha; como podemos observar, los negativos están a la izquierda y los positivos están a la derecha.

Tecleemos otro par de números, por ejemplo el 3 y el 3, vamos a estudiar lo que sucede si los números son iguales, los resultados que aparecen en pantalla son:

```
1      0
1      0
1      0
```

Observamos que los resultados NO, son los correspondientes a los operadores que no contienen el signo igual, es decir, <>, < y >, en cam-

bio, el resultado es SI para los operadores que contienen el símbolo igual (=, <=, >=).

De hecho esto nos pone de manifiesto unas propiedades de simetría importantes. Si observan los ejemplos que hemos realizado, siempre aparecen tres resultados NO y tres resultados SI. Esto no ocurre porque sí, sino que estos seis operadores se pueden agrupar por contrarios, es decir, en una situación cualquiera cuando el uno da NO el otro da SI y viceversa. En efecto:

1. El operador = es contrario a <>. O dos cosas son iguales o son distintas.

2. El operador >= es contrario al operador <. Es decir, o un número es mayor o igual que otro, o por el contrario es menor, ¡tome nota: sólo menor!

3. El operador <= es contrario al operador >. Es decir, o un número es menor o igual que otro, o por el contrario es mayor, ¡tome nota: sólo mayor!

4.3.2 La comparación de textos en BASIC

Los números y, por lo tanto, las variables numéricas forman parte de un conjunto ordenado que nos permite utilizar los operadores de comparación, ¿Es posible tratar los textos como un conjunto ordenado?

Vayamos por pasos: Consideremos, en primer lugar, textos de una letra. En este caso es fácil recordar el abecedario:

A,B,C,CH,D,E,F,G,H,I,J,K,L,M
N,Ñ,O,P,Q,R,S,T,U,V,X,Y,Z.

y decir que la letra F va antes que la L.

El abecedario es el que permite ordenar las letras

El abecedario nos muestra el orden de las letras, es posible a partir de este orden contestar la pregunta:

¿La letra F es mayor que la letra L?

con un SI o con un NO (en este caso es NO).

La misma pregunta pueda formularse con los otros operadores:

¿La letra F es mayor o igual que la letra L? NO.

¿La letra F es menor que la letra L? SI.

¿La letra F es menor o igual que la letra L? SI.

Podemos concluir que los textos de una letra forman un conjunto ordenado, cuya ordenación se establece mediante el abecedario.

En realidad, el mecanismo que nos ha llevado a establecer este orden es la asociación de cada letra al número de orden de ella misma en el

abecedario. En efecto, escribamos de nuevo el abecedario con el número de orden al lado:

01-A	06-E	11-J	16-N	21-R	26-W
02-B	07-F	12-K	17-Ñ	22-S	27-X
03-C	08-G	13-L	18-O	23-T	28-Y
04-CH	09-H	14-LL	19-P	24-U	29-Z
05-D	10-I	15-M	20-Q	25-V	

y planteemos la pregunta del modo siguiente:

¿El número asociado a la letra F es mayor que el número asociado a la letra L?

La pregunta se ha planteado finalmente como una comparación de números.

Si tomamos el abecedario que podrían utilizar unos habitantes de un planeta imaginario, con el siguiente orden de letras, por ejemplo:

01-M	06-LL	11-J	16-X	21-D	26-Ñ
02-CH	07-B	12-K	17-A	22-G	27-S
03-L	08-P	13-Q	18-E	23-Y	28-U
04-H	09-I	14-V	19-C	24-T	29-R
05-N	10-Z	15-W	20-F	25-O	

se ha definido, inmediatamente, un nuevo orden y la pregunta:

¿La letra F es mayor que la letra L?

tiene una respuesta positiva, es decir SI, lo contrario que ocurre en nuestro abecedario, porque en el nuevo orden de las letras la F está después de la L. Con este ejemplo, se quiere resaltar el tipo de acuerdo, la estandarización que significa el abecedario y que afortunadamente compartimos todos los que vivimos en la tierra, aunque con ligeras discrepancias como veremos más adelante.

Repetimos que el elemento más importante para establecer la comparación entre textos de una letra es:

— Ligar a cada letra un número mediante la definición del abecedario.

— Una vez hemos obtenido este número, los términos de comparación se establecen igual que en los números. Lo que hemos hecho es establecer un procedimiento de medida de tal manera que a cada letra le asignamos un número.

4.3.2.1 El orden lexicográfico

El abecedario aparte de resolver la comparación entre textos de una letra, nos abre el camino para poder comparar los textos de más de una letra. Para hacerlo, definamos la regla siguiente:

La comparación entre dos textos de más de una letra es igual a la comparación de la primera letra que no es igual en los dos textos, de izquierda a derecha. Si no hay letras desiguales los textos cumplen la igualdad.

Vamos a analizar esta regla con unos ejemplos:

1. Compararemos los textos «ABETO» y «CODO».

Las primeras letras, la A de ABETO y la C de CODO, son distintas.

La A es la primera letra del alfabeto y la C es la tercera, por lo tanto, la A es menor que la C, en consecuencia, el texto ABETO es menor que el texto CODO. Los resultados de aplicar todos los operadores relacionales entre estos dos textos sería:

TEXTO 1	OPERADOR	TEXTO 2	RESPUESTA
ABETO	es igual (=) que	CODO	NO
	es distinto (<>) que		SI
	es mayor (>) que		NO
	es mayor o igual (>=) que		NO
	es menor (<) que		SI
	es menor o igual (<=) que		SI

2. Comparamos los textos «CODO» y «CODO».

No hay letras distintas, luego los textos son iguales.
Los resultados de aplicar los diversos operadores es:

TEXTO 1	OPERADOR	TEXTO 2	RESPUESTA
CODO	es igual (=) que	CODO	SI
	es distinto (<>) que		NO
	es mayor (>) que		NO
	es mayor o igual (>=) que		SI
	es menor (<) que		NO
	es menor o igual (<=) que		SI

3. Comparemos los textos «GARCIA» y «GARZA».

Ignoramos las tres primeras letras, leídas de izquierda a derecha, porque son iguales (GAR). La primera letra distinta es la C en GARCIA y la Z en GARZA. Como la letra C es menor que la letra Z, por lo tanto, GARCIA es menor que GARZA.

Los resultados de aplicar los operadores es:

TEXTO 1	OPERADOR	TEXTO 2	RESPUESTA
GARCIA	es igual (=) que	GARZA	NO
	es distinto (<>) que		SI
	es mayor (>) que		NO
	es mayor o igual (>=) que		NO
	es menor (<) que		SI
	es menor o igual (<=) que		SI

El orden establecido mediante el abecedario y la regla que hemos enunciado para comparar textos de más de una letra se denomina orden lexicográfico.

Es interesante redactar unas instrucciones para establecer el orden lexicográfico de una manera más precisa que la regla. (Dese cuenta que vamos a escribir un programa.)

El seguimiento de un programa se realiza mediante líneas de memoria

Las instrucciones que se quieren escribir pretenden que dados dos textos y un operador relacional, se obtenga un resultado de SI o NO. Las figuras 1 y 2, muestran unas instrucciones para llevar a cabo una comparación lexicográfica. Las instrucciones tienen dos partes bien diferenciadas. La figura 1 describe los datos que vamos a utilizar, tanto los iniciales como los intermedios, como el resultado que se pretende alcanzar. La figura 2 contiene las instrucciones propiamente dichas. Antes de seguir adelante debe leer con atención las instrucciones de las dos figuras. Aunque le sea algo difícil de entender no se preocupe; a continuación le explicaremos muy detalladamente el funcionamiento.

Para entender cómo funcionan estas instrucciones las aplicaremos a los ejemplos anteriores.

Antes, sin embargo, observe la figura 3. En ella hay una tabla que simula una memoria para realizar la comparación:

— En la parte superior se ven los titulares de las distintas variables (TEXTO 1, OPERADOR, TEXTO 2, etc.).

— En la primera columna de la izquierda se le va indicando el paso realizado. (Paso 0, 1, 2, etc., hasta el paso 10). Lo que se hace en cada uno de estos pasos está descrito en la figura 2.

TEXTO 1	:	**Primer texto a comparar.**
OPERADOR	:	=
		<>
		<
		<=
		>
		>=
TEXTO 2	:	**Segundo texto a comparar.**
RESULTADO	:	**SI ó NO**

DATOS INTERMEDIOS

NUMERO DE ORDEN	:	**Número que indica qué letra del texto comparamos, es decir, la primera, la segunda, la tercera, etc.**
NUM1	:	**Número de orden, en el abecedario, de una letra perteneciente al primer texto.**
NUM2	:	**Número de orden, en el abecedario, de una letra perteneciente al segundo texto.**

Figura 1: Datos para la comparación lexicográfica

Paso 0. INICIO.

Paso 1. Sea el NUMERO DE ORDEN de la letra a considerar el uno.
(No confunda con la primera letra del abecedario, este número se refiere a la letra número uno de los textos.)

Paso 2. SI el TEXTO1 tiene una longitud mayor o igual que el NUMERO DE ORDEN actual.
ENTONCES
Coloque en NUM1, el número que la letra NUMERO DE ORDEN, en el TEXTO1 lleva adosada en el abecedario.
EN CASO CONTRARIO
Coloque en NUM1 un cero.

Paso 3. SI el TEXTO2 tiene una longitud mayor o igual que el NUMERO DE ORDEN actual.
ENTONCES
Coloque en NUM2, el número que la letra NUMERO DE ORDEN, en el TEXTO2 lleva adosada en el abecedario.
EN CASO CONTRARIO
Coloque en NUM2 un cero.

Paso 4. SI el NUM1 es cero ENTONCES IR al paso 9.

Paso 5. SI el NUM2 es cero ENTONCES IR al paso 9.

Paso 6. SI el NUM1 es distinto al NUM2 ENTONCES ir al paso 9.

Paso 7. Incrementar el NUMERO DE ORDEN en uno.

Paso 8. IR al paso 2.

Paso 9. El resultado es el de aplicar la comparación NUM1 OPERADOR NUM2

Paso 10. FIN.

Figura 2: Instrucciones para la comparación lexicográfica

Figura 3: Seguimiento de la comparación de ABETO y CODO

Paso	TEXTO1	OP.	TEXTO2	NUMERO DE ORDEN	NUM1	NUM2	RESULTADO
0	ABETO	<	CODO	—	—	—	—
1	ABETO	<	CODO	1	—	—	—
2	ABETO	<	CODO	1	1	—	—
3	ABETO	<	CODO	1	1	3	—
4	ABETO	<	CODO	1	1	3	—
5	ABETO	<	CODO	1	1	3	—
6	ABETO	<	CODO	1	1	3	—
9	ABETO	<	CODO	1	1	3	SI
10	ABETO	<	CODO	1	1	3	SI

— En cada una de las líneas, la primera columna indica el paso realizado y en el resto de la línea la situación de cada una de las variables hasta ese momento.

Comencemos a hacer el programa, siguiendo los pasos de la figura 2. Le advertimos que lo único que vamos a hacer en este programa es comparar la primera letra de los dos textos. Por tanto, todo el proceso que describimos a continuación servirá para comparar la A de ABETO y la C de CODO únicamente.

Paso 0: Es el inicio. Por eso, en la *línea 0* aparece la situación inicial del Problema. En el TEXTO1 está ABETO, en OP tenemos el operador «menor que» (<) y en TEXTO2 está CODO. Pretendemos, por tanto, realizar la comparación siguiente: ¿ABETO es menor que CODO? El resto de las variables de esta línea no se han utilizado aún, y por eso se indica con una raya.

Paso 1: En el paso 1 la instrucción es «sea el NUMERO DE ORDEN de la letra a considerar el uno»; es decir, se indica que la letra que se va a comparar es la primera (número 1) del texto. El resultado de esta instrucción es lo que aparece en la *línea 1,* en la que se ha colocado un 1 en la columna NUMERO DE ORDEN. Los demás datos no han variado.

Paso 2: En el paso 2 la instrucción indica lo que se debe hacer según que el TEXTO1 tenga mayor o menor número de letras que el NUMERO DE ORDEN en el que estamos en este momento. Puesto que el NUMERO DE ORDEN (el uno) es menor que la longitud del TEXTO1 (que tiene 5 letras), la instrucción ordena que ENTONCES coloque en la columna NUM1 el número que corresponde a la letra *A,* que es la primera de ABETO. Es lo que aparece en la *línea 2* de la figura 3. Las demás variables todavía no se han tocado.

Paso 3: La instrucción del paso 3 es similar a la del paso 2, pero referida al TEXTO2. Es decir, es comparar si el NUMERO DE ORDEN en el que estamos (el uno), es menor que la longitud del TEXTO2. Puesto que es menor, ENTONCES, debe poner en la columna NUM2 el número correspondiente a la letra *C* en el abecedario. Al ser la letra *C,* la tercera letra, se debe poner en esta columna un 3. Es lo que aparece en la *línea 3.* Como todavía no tenemos ningún resultado, aparece en esta columna una raya.

En las columnas aparece la memoria y en las filas los pasos del programa

Paso 4: En esta instrucción se indica que si en la columna NUM1 hay un cero que se pase al paso 9. Puesto que no tenemos un cero, sino un uno, seguimos con el paso siguiente. Por eso verá que la *línea 4* queda igual que la anterior.

Paso 5: Esta instrucción es similar a la anterior, pero referida a la variable NUM2; es decir, se ordena que si en la columna NUM2 hay cero que se vaya al paso 9. Puesto que hay un uno, se sigue el proceso. La *línea 5* no sufre variación.

Paso 6: La instrucción indica que si los números anotados en NUM1 y NUM2 son distintos se pase al paso 9. Por eso, la *línea 6* permanece igual, pero el programa salta al paso 9. En caso contrario hubiera seguido con el paso 7.

Paso 9: En la instrucción del paso 9 se indica que el resultado (SI o NO) sale de comparar NUM1 y NUM2. Por eso, en la *línea 9,* aparece la respuesta afirmativa en la columna RESUL., puesto que son distintos.

Paso 10: En el paso 10 se ordena acabar el programa. Por eso, la *línea 10* sigue igual.

Paso 7: Este paso se hubiera realizado si en el paso NUM1 y NUM2 hubieran sido iguales. En este caso, en la columna NUMERO DE ORDEN se hubiera aumentado en una unidad, y en la línea correspondiente habría aparecido en este caso un 2, puesto que el anterior era 1.

Paso 8: El paso 8 nos manda ir al paso 2, puesto que se ha finalizado la ejecución de todas las instrucciones. La *línea 8* no variaría respecto a la 7.

Observe que en los pasos 4, 5 y 6 las columnas de las variables no se han modificado.

El segundo ejemplo es la comparación de CODO con CODO para el operador distinto que (<>).

Antes de observar la figura 4 y leer los comentarios debería tomar un papel y un lápiz, construirse una tabla semejante al de la figura 3 y rellenarla siguiendo las instrucciones de la figura 2. Si no es capaz todavía de hacerlo, no se preocupe, pues a continuación le pondremos el proceso. De todas formas, inténtelo y no se dé fácilmente por vencido.

Como puede observar este procedimiento es más largo que en el ejemplo anterior, se debe a que las letras son iguales y aparecen cuatro repeticiones de los pasos 2, 3, 4, 5, 6, 7 y 8.

En el primer bloque de pasos del 2 al 8, el esquema de los pasos 2, 3, 4, 5 y 6 es igual al de la figura 3. La diferencia está en que NUM1 y NUM2 son ahora iguales, valen 3 pues corresponden a la tercera letra del alfabeto la C. Por lo tanto, después del paso 6 se ejecuta el paso 7 (en la figura 3 se ejecutaba el 9) que consiste en incrementar el NUMERO DE ORDEN a 2, es decir, pasamos a considerar la segunda letra de los textos ya que la primera es igual en ambos.

Paso	TEXTO1	OP.	TEXTO2	NUM1	NUM2	NUMERO DE ORDEN	RESULTADO	
0	CODO	<>	CODO	—	—	—	—	
1	CODO	<>	CODO	—	—	1	—	
2	CODO	<>	CODO	3	—	1	—	
3	CODO	<>	CODO	3	3	1	—	
4	CODO	<>	CODO	3	3	1	—	
5	CODO	<>	CODO	3	3	1	—	1A.
6	CODO	<>	CODO	3	3	1	—	LETRA
7	CODO	<>	CODO	3	3	2	—	
8	CODO	<>	CODO	3	3	2	—	
2	CODO	<>	CODO	18	—	2	—	
3	CODO	<>	CODO	18	18	2	—	
4	CODO	<>	CODO	18	18	2	—	
5	CODO	<>	CODO	18	18	2	—	2A.
6	CODO	<>	CODO	18	18	2	—	LETRA
7	CODO	<>	CODO	18	18	3	—	
8	CODO	<>	CODO	18	18	3	—	
2	CODO	<>	CODO	5	—	3	—	
3	CODO	<>	CODO	5	5	3	—	
4	CODO	<>	CODO	5	5	3	—	
5	CODO	<>	CODO	5	5	3	—	3A.
6	CODO	<>	CODO	5	5	3	—	LETRA
7	CODO	<>	CODO	5	5	4	—	
8	CODO	<>	CODO	5	5	4	—	
2	CODO	<>	CODO	18	—	4	—	
3	CODO	<>	CODO	18	18	4	—	
4	CODO	<>	CODO	18	18	4	—	
5	CODO	<>	CODO	18	18	4	—	4A.
6	CODO	<>	CODO	18	18	4	—	LETRA
7	CODO	<>	CODO	18	18	5	—	
8	CODO	<>	CODO	18	18	5	—	
2	CODO	<>	CODO	0	—	5	—	
3	CODO	<>	CODO	0	0	5	—	
4	CODO	<>	CODO	0	0	5	—	
9	CODO	<>	CODO	0	0	5	NO	
10	CODO	<>	CODO	0	0	5	NO	

Figura 4: Seguimiento de la comparación de CODO y CODO

En la figura 4 se marca una separación entre estos bloques, que únicamente difieren en el NUM1 y NUM2. Así, en el primero aparece el 3 de la C, en el segundo el 18 de la O, en el tercero el 5 de la D y en el cuarto el 18 de la O.

A partir del cuarto bloque se empieza con el paso 2 de las instrucciones considerando la quinta letra. Como no existe una quinta letra en el TEXTO1, es CODO, utiliza la parte de EN CASO CONTRARIO y coloca un cero en NUM1.

En el paso 3, la situación es igual a la del paso 2, pero utilizando TEXTO2 y NUM2.

En el paso 4 como NUM1 es cero sigue el paso 9 sin alterar ningún dato intermedio.

En el paso 9 se calcula el resultado de comparar O (NUM1) es distinto de O (NUM2) y se coloca el resultado NO.

El paso 10 finaliza las instrucciones.

En la figura 5, se muestra el seguimiento de la comparación entre GARCIA y GARZA con el operador mayor o igual que (>=). Es conveniente que lo intente realizar antes de ver la figura.

Se observa una estructura muy similar a la figura 4, pues aparecen tres bloques de pasos del 2 al 8 que corresponden a la comparación de las tres primeras letras de los textos que son iguales la G, la A y la R.

En el nuevo paso 2, después del tercer bloque, se coloca un 3 de la C en NUM1, pues la C es la tercera letra del abecedario. En el paso 3 se coloca un 29, que es el que corresponde a la Z en el abecedario.

Los pasos 4 y 5 no hacen más que seguir pues no se ha acabado ningún texto.

Y el paso 6, nos envía al paso 9 pues 3 (NUM1) es distinto de 29 (NUM2).

Paso	TEXTO1	OP.	TEXTO2	NUM1	NUM2	NUMERO DE ORDEN	RESULTADO	
0	GARCIA	>=	GARZA	—	—	—	—	
1	GARCIA	>=	GARZA	—	—	1	—	
2	GARCIA	>=	GARZA	8	—	1	—	
3	GARCIA	>=	GARZA	8	8	1	—	
4	GARCIA	>=	GARZA	8	8	1	—	
5	GARCIA	>=	GARZA	8	8	1	—	1A.
6	GARCIA	>=	GARZA	8	8	1	—	LETRA
7	GARCIA	>=	GARZA	8	8	2	—	
8	GARCIA	>=	GARZA	8	8	2	—	
2	GARCIA	>=	GARZA	1	—	2	—	
3	GARCIA	>=	GARZA	1	1	2	—	
4	GARCIA	>=	GARZA	1	1	2	—	
5	GARCIA	>=	GARZA	1	1	2	—	2A.
6	GARCIA	>=	GARZA	1	1	2	—	LETRA
7	GARCIA	>=	GARZA	1	1	3	—	
8	GARCIA	>=	GARZA	1	1	3	—	
2	GARCIA	>=	GARZA	21	—	3	—	
3	GARCIA	>=	GARZA	21	21	3	—	
4	GARCIA	>=	GARZA	21	21	3	—	
5	GARCIA	>=	GARZA	21	21	3	—	3A.
6	GARCIA	>=	GARZA	21	21	3	—	LETRA
7	GARCIA	>=	GARZA	21	21	4	—	
8	GARCIA	>=	GARZA	21	21	4	—	
2	GARCIA	>=	GARZA	3	—	4	—	
3	GARCIA	>=	GARZA	3	29	4	—	
4	GARCIA	>=	GARZA	3	29	4	—	
5	GARCIA	>=	GARZA	3	29	4	—	
6	GARCIA	>=	GARZA	3	29	4	—	
9	GARCIA	>=	GARZA	3	29	4	NO	
10	GARCIA	>=	GARZA	3	29	4	NO	

Figura 5: Seguimiento de la comparación de GARCIA y GARZA

En el paso 9 se calcula el resultado de comparar 3 es mayor o igual que 29 que tiene como respuesta NO, tal como aparece en la columna de resultado.

Aparte de ilustrar los ejemplos, el método del seguimiento es muy útil para comprobar la corrección de los programas. En este apartado se ha de introducir para comprobación de unas instrucciones; más adelante veremos cómo se aplica en el caso de programas.

Le recomendamos que repita el seguimiento con los ejemplos dados que adquiera una cierta soltura. Este ejercicio le será de gran utilidad para irse acostumbrando al análisis informático. Incluso, puede ponerse usted mismo otros ejemplos, una vez que tenga asimilados estos que le proponemos.

4.3.2.2 Algunos casos particulares

Antes de cerrar los temas de las comparaciones con textos, es necesario reflexionar un poco sobre el abecedario que utiliza el ordenador.

El código ASCII es el abecedario del ordenador

El alfabeto que utiliza un ordenador es generalmente el código ASCII, como se havisto en el capítulo anterior aunque ciertamente no es el único, es sin embargo, con mucho el más extendido.

La reflexión debe hacerse en los puntos siguientes:

1) Letras dobles:

Las letras dobles

En el abecedario castellano existen dos letras dobles: la LL y la CH; se llaman dobles porque consumen dos símbolos para expresarse.

Si consideramos el resultado de comparar los textos

LLOBET Y LOPEZ

mediante el criterio lexicográfico, utilizando el alfabeto castellano, la primera diferencia que observamos es precisamente en la primera letra, en LLOBET es la LL y en LOPEZ es la L, como la L es anterior a la LL el resultado es que LLOBET es mayor que LOPEZ.

Si esta misma comparación se realiza con el código ASCII como abecedario, al no existir letras dobles la primera diferencia aparece con la segunda letra, la L de LLOBET y la O de LOPEZ. Como la O es mayor que la L, el resultado nos dice que el texto LLOBET es menor que el texto LOPEZ. Resultado contradictorio con el obtenido por comparación con el alfabeto castellano.

De manera general el código ASCII, y los demás códigos usados por los ordenadores, no tienen en cuenta la posibilidad de utilizar dos símbolos para las comparaciones.

Esta propiedad no es únicamente para el castellano, existen muchos idiomas que utilizan letras dobles.

2) La letra Ñ.

Los caracteres especiales en los diversos lenguajes

Si observamos el código ASCII (consulte la figura 6 de la lección 3), veremos que la letra Ñ no aparece. La razón es que en el idioma inglés no

existe esta letra y el código ASCII se diseñó en países de habla inglesa, que olvidaron las particularidades de los otros idiomas:

Actualmente, debido sin duda a la expansión del mercado de los ordenadores, hay muchos dispositivos que incorporan esta letra en el repertorio de teclados, pantallas e impresoras.

Entonces se puede decir que el ordenador no funciona con el código ASCII, o mejor dicho, que funciona con el código ASCII modificado.

La figura 6 nos muestra un cuadro comparativo de estas modificaciones. En la columna de la ízquierda indica los diferentes idiomas, en las restantes columnas se indica el número del carácter ASCII que se modifica. La primera fila que corresponde a USA (Estados Unidos), es el símbolo correspondiente al código ASCII según su definición, es decir, sin modificar.

Pero volviendo al hilo de la cuestión, observamos que la letra Ñ en el código ASCII español está situada en la posición 92. Esto quiere decir que esta posición en la comparación del texto:

ÑANDU y ZULU

dan a ZULU como menor que ÑANDU, en contradicción flagrante con el orden establecido en el diccionario.

En los demás signos alterados observaríamos efectos parecidos, sin embargo con mucho el más significativo es el de la Ñ.

3) La comparación de letras minúsculas y mayúsculas.

Las letras mayúsculas y minúsculas

La observación de la figura 6 del capítulo 3 nos muestra que todas las letras mayúsculas están delante de las minúsculas; esto en los operadores de comparación presenta algunas dificultades, y más que dificultades, contradicciones respecto a lo que esperamos. Si uno espera una cosa y sucede otra, las consecuencias pueden ser muy graves para un programa; el paso de funcionar a no funcionar es muy pequeño.

Aclaremos estos conceptos con varios ejemplos:

a) Si comparamos *ALA* y *Ala*, el resultado es distinto pues para el ordenador no es lo mismo las letras mayúsculas y minúsculas. Por ser la letra L menor en el código ASCII que la letra l minúscula, la palabra ALA se

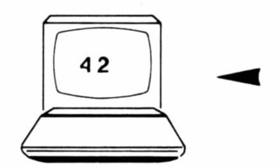
	Números del Código ASCII										
	35	64	91	92	93	94	96	123	124	125	126
U.S.A.	#	@	[	/	]	^	`	{	¦	}	~
Inglaterra	£	@	[	/	]	^	`	{	¦	}	~
Alemania	#	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
Dinamarca	#	@	Æ	Φ	Å	^	`	æ	Ø	å	~
Francia	£	à	·	ç	§	^	`	é	ù	è	¨
Suecia	#	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü
Italia	#	§	·	ç	é	^	ù	à	ò	è	ì
España	#	@	¡	Ñ	¿	^	`	¨	ñ	}	~

Figura 6: Cuadro comparativo de los códigos ASCII modificados

coloca antes que la palabra Ala, o sea la palabra ALA es menor que la palabra Ala.

b) Si comparamos *AZADA* con *Ala,* la primera diferencia en el orden lexicográfico aparece en la segunda letra, en una es una Z y en otra es una l. Al ser la l minúscula, su código ASCII es mayor que el de la letra Z. En consecuencia la palabra AZADA es menor que la palabra Ala.

Mientras el problema de la letra doble y de la ñ no tienen prácticamente solución, en las de las minúsculas existen soluciones que veremos más adelante. Sin embargo, conviene insistir que en la tarea de programación estas particularidades hay que tenerlas siempre presentes.



4.4 LA CODIFICACION

Antes de acabar con los operadores relacionales hemos de considerar un tema que tocamos ya de pasada, la representación numérica de pocas alternativas.

Las únicas posibilidades como respuesta de las preguntas binarias eran el SI o el NO. Para poder manipularlas en el ordenador las transformábamos en los números siguientes, para el NO el 0 y para el SI, el 1 o el −1, según el ordenador.

A esta operación de transformar diversas alternativas en un número se denomina codificación.

También en esta lección hemos hablado del código ASCII, que es otro ejemplo de codificación. Cada símbolo del alfabeto es codificado según un número, de modo que las variables textuales en el interior del ordenador no son más que números. Por ejemplo, la palabra GARCIA en el interior del ordenador es la secuencia de números 71(G), 65(A), 82(R), 67(C), 73(I), 65(A). Consulte la tabla del código ASCII en la figura 6 de la lección 3.

El método de asignar un número a cada alternativa se denomina codificación, de aquí el nombre de código ASCII.

La ordenación es una propiedad importante a tener en cuenta al escoger un código.

Hemos mencionado también que la elección del código ASCII tenía unas consecuencias en la comparación de las variables textuales. Cada vez que usemos el método de codificación debemos tener presente estas consecuencias respecto a la ordenación.

Un ejemplo de codificación es la utilización de los días de la semana para hacer cálculos en el ordenador. Como sabemos los días de la semana son 7, LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO y DOMINGO. La forma más natural de trabajar con los días de la semana en el ordenador es asociarles un número del 0 al 6 (la inclusión del 0 significa 7 alternativas) del modo siguiente:

Alternativa	Código
LUNES	0
MARTES	1
MIERCOLES	2
JUEVES	3
VIERNES	4
SABADO	5
DOMINGO	6

En el programa trabajaremos con estos números en lugar de con los nombres de los días de la semana. Estos números son los códigos que representan los días de la semana.

La elección de estos números ha sido arbitraria; podíamos elegir otros números cualesquiera; sin embargo, la elección de este código presenta ventajas que vamos a estudiar a continuación.

La utilización de estos números en operaciones aritméticas puede ser peligroso. En efecto, si sumamos tres días al viernes, cuyo código es 4, obtenemos un 7, es decir, obtenemos un código que no corresponde a ningún día de la semana. Debemos estar alerta, por lo tanto, que en los cálculos de nuestro programa mantengamos los códigos a valores que realmente tengan significado.

Para profundizar más en este ejemplo, pensemos en realizar un programa que, dado un día de la semana, nos calcule el código del día siguiente. Para ser más precisos, tenemos una variable numérica DS (Día de la Semana) que contendrá el código de un día de la semana y queremos calcular la variable SS (día Siguiente de la Semana), que nos da el código del día siguiente de la semana. Un ejemplo de este programa es:

```
NEW
10 INPUT DS :REM Se entra el codigo
                del dia de la semana.
20 LET SS= MOD ((DS+1), 7): REM Calculo
                del codigo del dia siguiente.
30 PRINT SS
40 GOTO 10
```

El núcleo del programa es la instrucción 20. La operación MOD ((DS+1),7) es precisamente la que impide que el código resultante sobrepase el valor 6. En efecto, consideremos los posibles valores del código. Los resultados posibles se muestran en la tabla que sigue a continuación: (Puede ir dándole a la variable DS los valores de la tabla y comprobará que el programa le da los resultados.)

Día Semana	DS	DS+1	RESTO de DIVIDIR POR 7	SS(RESULTADO)
LUNES	0	1	1	1 MARTES
MARTES	1	2	2	2 MIERCOLES
MIERCOLES	2	3	3	3 JUEVES
JUEVES	3	4	4	4 VIERNES
VIERNES	4	5	5	5 SABADO
SABADO	5	6	6	6 DOMINGO
DOMINGO	6	7 ---->>	0	0 LUNES

Como puede observarse, el comportamiento de la fórmula es muy correcto, la gracia está en utilizar la función MOD que nos asegura siempre un resultado entre 0 y 6.

Probablemente piense qué ocurriría, si se entrara un número en DS mayor que 6. La respuesta es que prácticamente las cosas funcionarían bien si tenemos la precaución de considerar que el 7 es el código también de un LUNES, el 8 el código de un MARTES, etc. De hecho la utilización de la función MOD nos previene incluso contra posibles errores en la entrada de los códigos de los días de la semana.

De todos modos, la belleza de estas fórmulas no han sido frutos de la casualidad. Si consideramos otro código para los días de la semana como el siguiente:

Alternativa	Código
LUNES	5
MARTES	2
MIERCOLES	1
JUEVES	0
VIERNES	3
SABADO	4
DOMINGO	6

Este nuevo código es perfectamente posible, porque tenemos la libertad de elegirlo. Sin embargo, todas las posibilidades de cálculo del día siguiente se han esfumado. En efecto, si reconstruimos la tabla resultante después de aplicar el programa, observaremos lo siguiente:

Día Semana	**DS**	**DS+1**	**RESTO de DIVIDIR por 7**	**SS**	
LUNES	5	6	6	6	DOMINGO
MARTES	2	3	3	3	VIERNES
MIERCOLES	1	2	2	2	MARTES
JUEVES	0	1	1	1	MIERCOLES
VIERNES	3	4	4	4	SABADO
SABADO	4	5	5	5	LUNES
DOMINGO	6	7 ---->>	0	0	JUEVES

Este mal comportamiento, se debe a que en la elección del segundo código hemos olvidado una propiedad muy importante de los días de la semana. Los días de la semana están ordenados, es decir, un LUNES siempre va antes de un MARTES, un MARTES va siempre antes de un MIERCOLES y así sucesivamente.

El primer código que hemos elegido respeta la ordenación de los días de la semana, el segundo no la respeta.

En consecuencia, cuando codifiquemos cualquier cosa, tarea muy frecuente cuando se trabaja con ordenadores, es necesario tener presente las propiedades de ordenación que debe tener el código. En caso contrario, los programas seran difíciles de realizar pues las propiedades del código no coinciden con las propiedades numéricas con las que trabaja el ordenador.

No todos los códigos tienen propiedades de ordenación

No todos los códigos, sin embargo, presentan estas propiedades de ordenación; entonces, buscar un orden es intentar asociar unas propiedades que el objeto que codificamos no tiene. Por ejemplo, en el juego del ajedrez los movimientos de la torre son cuatro:

ARRIBA

ABAJO

IZQUIERDA

DERECHA

Si tenemos que codificarlos, estamos construyendo, por ejemplo, un programa para jugar al ajedrez, podemos elegir cualquier código.

Como los movimientos de ARRIBA, ABAJO, IZQUIERDA y DERECHA no están ordenados, es decir, un movimiento no va antes que otro, la elección del código es mucho más simple.

Las normas generales que deben seguirse en todo proceso de codificación son:

1) Hacer una lista de todas las alternativas posibles.

2) Estudiar si estas alternativas deben ir ordenadas. En el caso de que así sea, ordenarlas de menor a mayor.

3) Asignar números correlativos a los distintos elementos, empezando por cero. La utilización del cero para el primer elemento no es obligatoria pero es recomendable.

En la práctica, la operación de codificación es sumamente importante para la programación. La operación es delicada y hay que estudiarla detenidamente antes de proceder a la codificación de unas alternativas; toda la sencillez o la dificultad de un programa pueden depender de este código.

RESUMEN

Cuando un conjunto de objetos tiene las propiedades de orden, se pueden aplicar los operadores de comparación, mayor, mayor o igual, menor, y menor o igual.

El conjunto de objetos más representativo que tiene propiedades de ordenación son los números. Los números, por otra parte, son las entidades que manipula el ordenador.

Cualquier conjunto de objetos a los que pueda asociarse un número mediante un proceso de medida se podrá someter a las preguntas de comparación.

Hay procesos de medida que no existen, sobre todo para variables de tipo psicológico, tales como el color, el sabor, el dolor, etc.

Este número debe ser único. Si al proceso de medida se le pueden asignar varios números a la vez, seguramente no se pueden hacer las preguntas. La posición de un barco es un ejemplo típico de medida con dos números.

En BASIC existen cuatro símbolos para cada uno de los operadores de comparación; el símbolo >= para el mayor o igual que, el símbolo > para el mayor que, el símbolo <= para el menor o igual que, y el símbolo < para el menor que.

Estos símbolos son únicos como grupo, de tal manera que <= es una entidad distinta a < y a =, del mismo modo que son entidades distintas INPUT y PRINT, aun cuando comparten las letras I, N, P y T.

Los números negativos son menores que los positivos. Dentro de los números negativos, los que tienen mayor valor absoluto son los más pequeños, es decir, el −10 es más pequeño que el −1.

También los operadores de comparación son contrarios, el mayor o igual que, tienen como contrario el menor que; y el menor o igual que tiene como contrario el mayor que. Operadores contrarios son aquellos que cuando uno da un SI como respuesta, el otro da un NO, y viceversa.

En la comparación con variables textuales se establece el orden de comparación mediante el número de orden en el abecedario para dos letras. El abecedario que utiliza el ordenador es el código ASCII.

Una letra es mayor o menor que otra según su código ASCII sea mayor o menor. La regla para comparar textos de más de una letra consiste en ignorar de izquierda a derecha las que son iguales, y entonces comparar las dos primeras letras distintas según el código ASCII. Si no hay letras distintas se le denomina orden lexicográfico. El orden lexicográfico presenta el inconveniente de no distinguir letras dobles, tales como la LL y la CH en castellano.

El código ASCII tomado como abecedario, tiene el inconveniente de que las letras mayúsculas tienen siempre un código menor que las letras minúsculas.

Es necesario tomar el código ASCII modificado para recoger las particularidades de cada idioma, en concreto, en el castellano la Ñ y las letras acentuadas se obtienen suprimiendo ciertos códigos en la definición del ASCII original.

En los ejemplos de comparación mediante el orden lexicográfico se utiliza un método de seguimiento de programas muy importante para depurar los programas informáticos. Consiste en hacer una tabla de las distintas variables y datos que intervienen en el problema y analizar cómo van variando a medida que se van ejecutando pasos del programa.

La codificación es una operación importante para representar numéricamente pocas alternativas. El propio SI o NO como respuesta a una pregunta binaria es un ejemplo de codificación. El código ASCII también es un ejemplo.

Las reglas para codificar correctamente son:

a) Listar las alternativas posibles

b) Ordenarlas si poseen la propiedad de orden

c) Asignar números correlativos, a ser posible empezando por el cero

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

26. La propiedad de .... es necesaria para poder aplicar los operadores de comparación.

27. Los .... constituyen el conjunto ordenado más representativo.

28. Mediante la .... se asigna un número a cada una de las alternativas.

29. La operación de codificación debe respetar el .... si el conjunto es ordenado.

30. Los símbolos que utiliza el ordenador para los operadores de .... son <=, <, >=, >.

31. En informática, para poder comparar hay que reducir a .... lo que se quiere comparar.

32. Cada operador relacional tiene su ....

33. Para comparar textos se utiliza como .... el código ASCII.

34. Cuando se comparan textos se utiliza el orden ....

35. El orden lexicográfico no tiene en cuenta las letras ....

Encierre en un círculo las respuestas que correspondan a la alternativa correcta:

36. De las cuatro parejas de operadores que se listan, ¿en qué pareja no aparecen operadores contrarios?

    a) <>, =

    b) <=, >=

    c) <=, >

    d) <, >=

37. La palabra COLIBRI es mayor que la palabra CHAPUZA porque en la comparación lexicográfica no se aplican las características de que:

    a) Las letras mayúsculas son iguales que las minúsculas

    b) La CH es una letra doble en el alfabeto castellano

    c) La Ñ es un símbolo especial en el código ASCII

    d) Las letras acentuadas tienen un código especial en el código ASCII

38. Qué regla no se aplica en la codificación:

    a) Listar las alternativas posibles

    b) Suprimir las alternativas incómodas

    c) Si tienen orden, ordenar las alternativas

    d) Asignar los números correlativamente

39. El seguimiento de programas en informática, se lleva a cabo haciendo:

    a) Una tabla de variables y datos

    b) Siguiendo paso a paso lo que ocurre

    c) Combinando las dos anteriores en una tabla

    d) Volviendo a escribir las instrucciones

40. Cuál de las siguientes comparaciones no es afirmativa:

    a) «30» <> «7»

    b) «30» > «7»

    c) 30>7

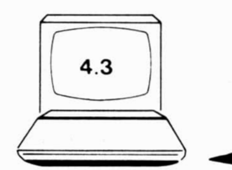
    d) 30>=7

41. Ordene la lista de números que se da a continuación, de modo que la aplicación entre dos elementos consecutivos de la pregunta ¿es este elemento menor o igual que el siguiente? tenga respuesta afirmativa:

    3, −5, 0, −0, 2

42. Ordene la lista de textos que damos a continuación, de modo que la aplicación entre dos elementos consecutivos de la pregunta ¿es este elemento mayor que el siguiente? tenga una respuesta afirmativa, utilizando el código ASCII:

    «agujero», «Paleta», «Zancadilla», «añorado», «azada»



43. Ordene los siguientes textos por orden lexicográfico de menor a mayor, utilizando el código ASCII (Recuerde que son textos, aunque tengan significado numérico):

    «1», «2», «10», «23», «120», «200»

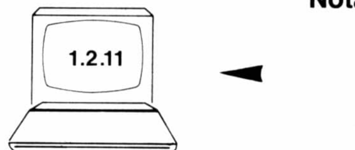
SOLUCIONES DE LOS EJERCICIOS DE AUTOCOMPROBACION

Capítulo 1

1. c
2. b
3. d
4. La figura debe corresponderse con la figura 6 de esta lección
5. b
6. d
7. b
8. 8.1 TECLADO: Entrada
   8.2 IMPRESORA: Salida
   8.3 CASSETTE: Entrada y Salida
   8.4 PANTALLA TV: Salida
9. Programación
10. Nivel
11. BASIC
12. Lectura
13. Pantalla
14. Multiplicación
15. Punto
16. Variables
17. Dólar
18. LET
19. F [El verbo (LET) debe ir al inicio de la instrucción]
20. F (A una variable textual no pueden asignárseles resultados numéricos)
21. V
22. V
23. F (Con las variables textuales no pueden realizarse operaciones aritméticas)
24. V
25. V
26. V
27. F (No pueden hacerse operaciones aritméticas con variables textuales)
28. F (No puede haber dos verbos en una instrucción)
29. V
30. F (No puede haber dos verbos en la misma instrucción)
31. V
32. V
33. F (8$ es un nombre incorrecto de variable)
34. Instrucción
35. Sentencia
36. Ejecutar
37. LIST
38. Número
39. Borra
40. LIST
41. Borra
42. RUN

43. INPUT
44. V
45. F (Sólo pueden ser números positivos)
46. F (El orden de ejecución sólo depende del número de sentencia)
47. V
48. F (Ejecuta un programa, no lo lista)
49. V
50. V
51. F (Permite entrar cualquier tipo de variable)
52. F (El espacio depende de la cantidad de sentencias, no de su numeración)
53. F (El comando RUN sirve para ejecutar un programa)
54. F (Borra la sentencia que había)
55. V
56. V
57. V
58. V
59. a) y el resultado es 18.8496
60. Las modificaciones pueden ser las siguientes:
    25 LET A = 3.1416 × R × R
    50 PRINT «EL AREA DE LA CIRCUNFERENCIA ES»
    60 PRINT A

Nota: El número de sentencia 25 puede ser 21, 22, 23... 29 y la variable A puede ser cualquier otra variable, excepto R y L.



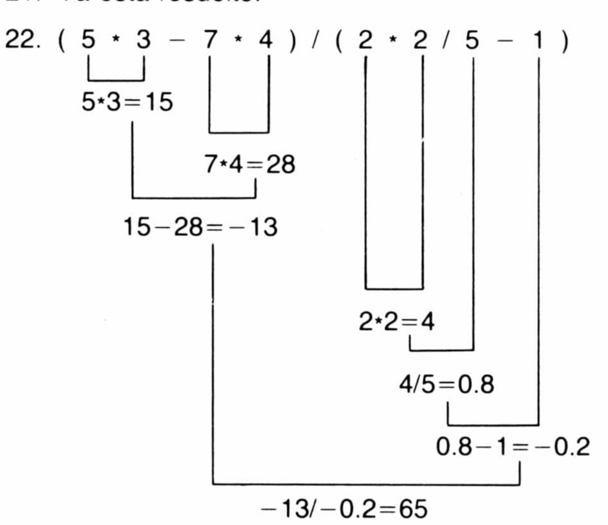
Capítulo 2

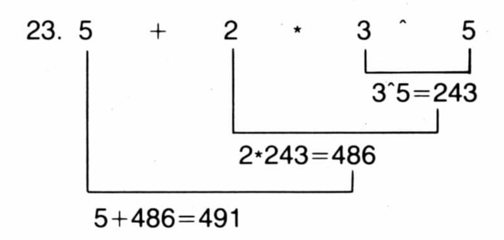
1. REM
2. instrucciones
3. PRINT
4. separadores
5. coma
6. coma
7. punto y coma
8. TAB (Encolumnador TAB)
9. columna
10. acumula
11. a
12. c
13. b
14. d
15. b
16. b
17. c
18. c
19. c
20. a
21. INPUT
22. lista
23. coma
24. BASIC
25. variables
26. SI

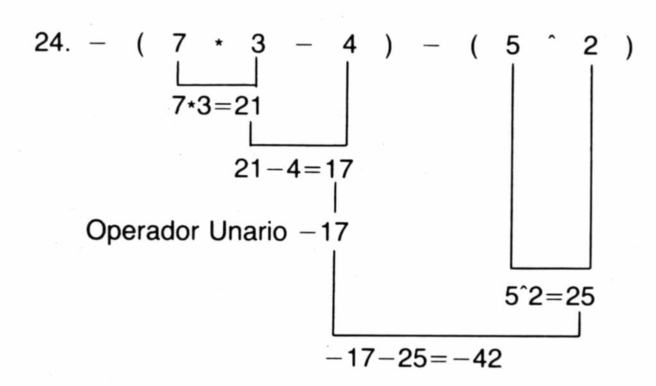
27. ignora
28. error
29. comillas
30. numéricas
31. c
32. b
33. a
34. d
35. b
36. b
37. b
38. a
39. a
40. d
41. V
42. V
43. V
44. F
45. V
46. V
47. F
48. F
49. V
50. V

Capítulo 3

1. Potenciación.
2. Binario.
3. Binario.
4. Un.
5. Unario.
6. Expresión.
7. Base.
8. Potenciación.
9. Realizar.
10. Exponente.
11. Paréntesis
12. Operadores
13. Más.
14. Balanceados.
15. Binaria.
16. Carácter.
17. Juntar.
18. Propiedad.
19. Comillas.
20. Vacío.
21. Ya está resuelto.
22. (5 * 3 − 7 * 4) / (2 * 2 / 5 − 1)



23.


24.


25. 7 – 5 * 3 + 6 ^ 2

5*3=15

7–15=–8

6^2=36

–8+36=28

26. c

27. d

28. a $(1\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0)$

29. b $(1\times2^5 + 1\times2^4 + 1\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0)$

30. b

31. c

32. a

33. b

34. c

35. d

36. unario

37. argumentos

38. argumentos
39. numéricos
40. resultado
41. comas
42. expresión
43. error
44. incorporarse
45. resultado
46. matemático
47. contrario (inverso)
48. ABS
49. SQR
50. fracción
51. LEN
52. trozos (partes)
53. inversas
54. STR$
55. PRINT
56. b
57. a
58. b
59. d
60. c
61. b
62. d
63. a
64. c
65. d
66. b
67. c

Capítulo 4

1. binarias
2. alternativa
3. sucesión
4. igualdad
5. cualidad
6. = y <>
7. tipo
8. instrucción y posición
9. numéricos
10. 1 y −1 (o viceversa)
11. *a*)
12. *d*)
13. *a*) La contestación tiene más de dos alternativas
14. *b*)
15. *b*) No se trata de una propiedad o cualidad medible
16. N
17. S
18. N
19. N
20. N
21. N
22. S
23. N
24. N
25. S
26. orden
27. números
28. codificación
29. orden
30. comparación
31. números
32. contrario
33. abecedario

34. lexicográfico
35. dobles
36. *b*) No pueden ser contrarios, si ambos contienen un igual
37. *b*)
38. *b*)
39. *c*)
40. *b*) Aunque los textos tengan significado numérico debe aplicar el orden lexicográfico, por lo tanto, «30»<«7». Es decir, al comparar 3 y 7, el 3 es menor que el 7.
41. −5,0,−0,2,3 El orden de los dos ceros (positivo y negativo) es indiferente pues el cero es independiente del signo
42. añorado, azada, agujero, Zancadilla, Paleta. Recuerde que la ñ es la letra que tiene mayor código ASCII, por tanto la palabra añorado es mayor que la palabra azada. También las letras minúsculas tienen mayor código que las mayúsculas
43. «1», «10», «120», «2», «200», «23» Aunque estos textos tengan significado numérico, lo está utilizando como textos y por lo tanto, debe seguir el orden lexicográfico

Parte II

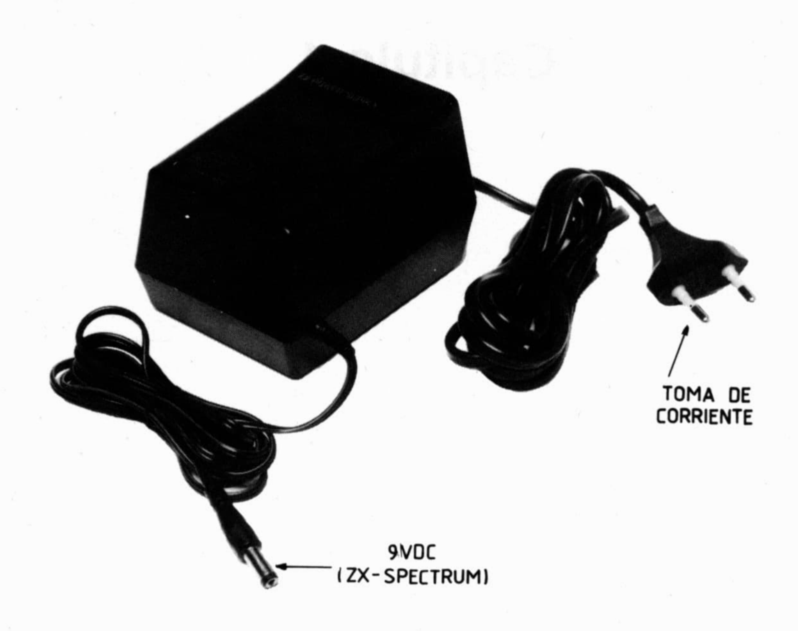
PRACTICAS CON EL ORDENADOR

Capítulo 1

ESQUEMA DE CONTENIDO

Instalación	Conexión y puesta en marcha
Teclado	Tecla FIN DE LINEA Modos de funcionamiento Teclas CAMBIAR MAYUSCULAS y FIJAR MAYUSCULAS Tecla BORRAR Tecla CAMBIAR SIMBOLO Teclas numéricas Espaciador Líneas con errores Numeración programas Resumen de los modos de operación Prácticas

Figura 1. Alimentador de corriente



1.1 INSTALACION

En la caja en la que se suministra el ordenador ZX-SPECTRUM se contienen los siguientes elementos:

1. Ordenador ZX-SPECTRUM.

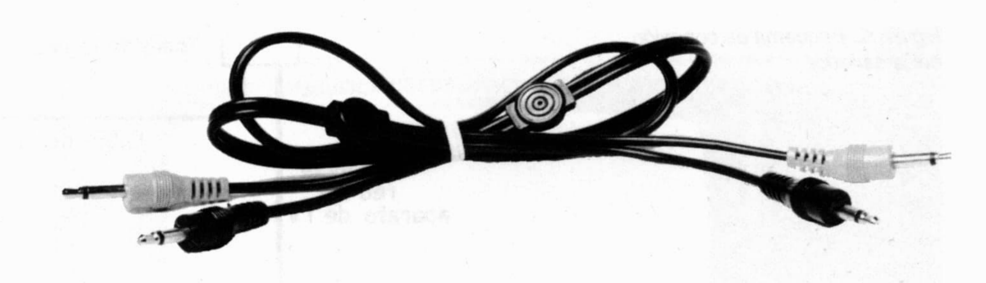
2. Una fuente de alimentación. Este aparato sirve para adaptar la corriente de la red a la que necesita el ordenador. (Fig. 1). Le advertimos que la fuente de alimentación que se suministra con el ordenador es para una tensión de 220 V. Caso de tener usted 125 debe proveerse de un transformador de 125 a 220 V.

3. Tres cables de conexión. Uno de ellos lo utilizaremos para conectar el televisor (Fig. 2). Los otros dos que van unidos servirán para conectar el ordenador a la grabadora de cassette (Fig. 3), cuando llegue el momento de utilizarla.



Figura 2. Cable de conexión con el televisor

Figura 3. Cable de conexión con la grabadora



Características del televisor

Para poder utilizar el ZX-SPECTRUM deberemos disponer de un televisor. La utilización de la grabadora de cassette la veremos en capítulos posteriores. El único requisito que debe cumplir el aparato de televisión es disponer de UHF. El ZX-SPECTRUM funciona en color. No hay ningún problema si el aparato de televisión es en blanco y negro. En este caso, los colores aparecerán como tonos de gris. Sin embargo, el funcionamiento del ordenador es idéntico en ambos casos.

1.1.1 Conexión y puesta en marcha

En la parte trasera del ordenador encontraremos los siguientes conectores hembra (Fig. 4):

9V DC

EAR

MIC

TV

Además, existe otro conector especial de forma alargada donde se pueden conectar dispositivos especiales. No hay ninguna posibilidad de conectar los cables de forma incorrecta puesto que todos los conectores tienen formas distintas y sólo encajan en el conector hembra correspondiente. En todo caso, vea las indicaciones de las figuras 1 y 2 que especifican dónde se ha de conectar y qué se debe conectar en este momento.

Cómo hacer la conexión

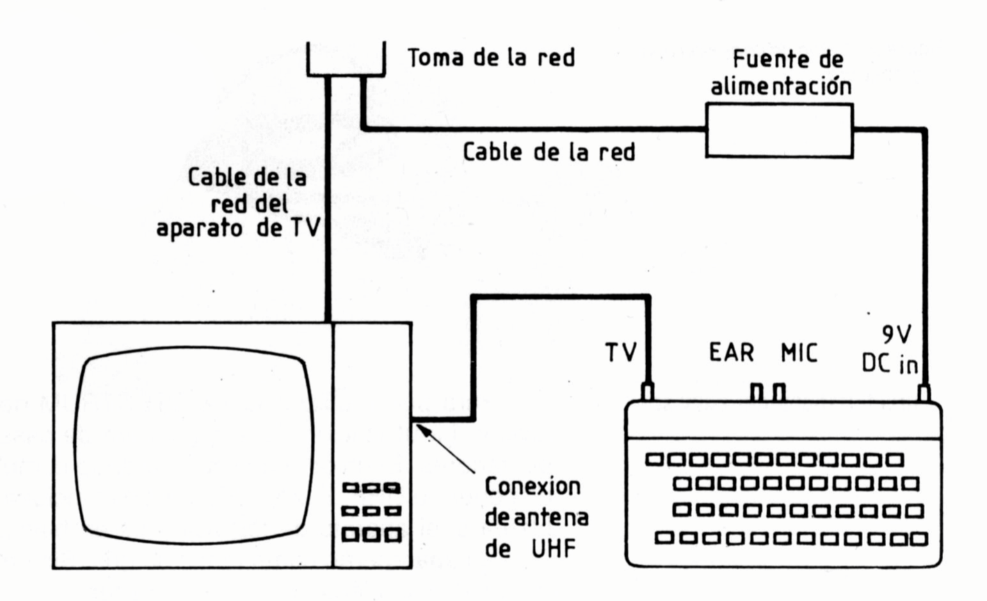
Los pasos a seguir para poner en marcha el ordenador son los siguientes:

1) Al aparato de televisión que vamos a utilizar, le desconectaremos el cable de antena si lo tuviera. A continuación, se conectará aquí el cable suministrado con el microordenador. (Ver figura 2.)

9VDC MIC EAR TV

Figura 4 Esquema de las conexiones en la parte posterior del ordenador.

Figura 5. Esquema de conexión del ordenador



2) El otro extremo del cable se conectará en el conector marcado TV en la parte trasera del ZX-SPECTRUM.

3) Conectaremos la fuente de alimentación a la toma de red. Comprobaremos antes que el voltaje (220 V o 125 V) sea el apropiado. La fuente de alimentación que se suministra con el microordenador es para 220 V. Caso de tener usted 125 V debe proveerse de un transformador.

4) La fuente de alimentación tiene otro cable que se conectará en el conector marcado 9V DC en la parte posterior del ZX-SPECTRUM. En la figura 5 tiene un esquema de todas estas conexiones.

5) Encenderemos ahora el televisor poniendo el mando del volumen al mínimo. En pantalla aparece la típica «nieve» que indica que no hay sintonía.

6) Hay que realizar ahora el proceso de sintonizado. El ZX-SPECTRUM trabaja en la banda de UHF y por tanto debemos seleccionar esta banda en el televisor. Si el aparato de televisión que usamos dispone de un mando de sintonía graduado, lo moveremos hasta el canal 36. Si por el contrario, el mando de sintonía no tiene graduación, deberemos girarlo lentamente hasta conseguir sintonizarlo. Si el aparato de televisión tiene un botón para cada canal, escogeremos uno no utilizado y giraremos el botón de sintonía correspondiente a dicho canal (recordemos que debe estar en la banda de UHF).

Esta operación es algo laboriosa la primera vez, pero con un poco de paciencia se efectúa sin mayores dificultades.

Cuando hayamos sintonizado el canal, aparecerá en pantalla el mensaje que se ve en la figura 6. Es conveniente apuntar en un papel la posición del mando o incluso, si es posible, hacer una pequeña marca con un rotulador sobre el mando. De esta forma, las próximas veces será más fácil sintonizarlo.

Figura 6. Mensaje que aparece al sintonizar el televisor



1.2 TECLADO

El teclado del ZX-SPECTRUM tiene básicamente la misma distribución que el teclado de una máquina de escribir. Comprobamos inmediatamente que, además, cada tecla sirve para varias cosas a la vez. Dependiendo de la situación (o modo de operación) una tecla escribirá una cosa u otra.

Modos de operación

Por ejemplo, si tomamos la tecla correspondiente a la letra K, vemos que con ella podemos hacer las siguientes cosas:

1) Púlsela directamente; aparecerá en pantalla la palabra LIST.

2) Púlsela de nuevo; aparecerá la letra K.

3) Pulse la tecla SYMBOL SHIFT, que se encuentra en la parte inferior del teclado y simultáneamente pulse la letra K; aparecerá en pantalla el signo +.

4) Pulse la tecla CAPS SHIFT, junto con la SYMBOL SHIFT. Si ahora pulsa de nuevo la letra K, aparecerá en pantalla la palabra LEN.

5) Pulse la tecla CAPS SHIFT, junto con la SYMBOL SHIFT de nuevo. Pulse ahora la tecla SYMBOL SHIFT y al mismo tiempo la tecla K. Aparecerá en pantalla la palabra SCREEN$.

Ciertamente el teclado del ZX-SPECTRUM es algo sofisticado y por ellos lo iremos aprendiendo poco a poco. Si al probar alguna tecla nos vemos en dificultades que no sabemos resolver, desconectaremos el conector 9V DC durante unos instantes para que se borre toda la memoria y empezaremos de nuevo.

1.2.1 Tecla FIN DE LINEA

En su ordenador esta tecla lleva escrita la palabra ENTER (entrar en inglés), situada en la parte derecha del teclado. Pulsándola, desaparece el mensaje inicial de la pantalla. En su lugar, aparece un recuadro en la parte inferior izquierda. Este recuadro tiene en su interior una letra intermitente El recuadro recibe el nombre del cursor. La letra de su interior nos indica el modo de operación en que se encuentra el teclado. No se preocupe ahora por esto de los modos de operación. Más adelante relacionaremos todos los modos de operación existentes con su correspondiente letra indicativa. Si usted pulsa en este momento la tecla ENTER, aparecerá la letra K en intermitencia. ¡Hágalo!

La letra del cursor

La misión fundamental de la tecla ENTER es la de informar al ordenador de que hemos terminado una línea. Si escribimos algo, el ordenador se queda esperando a que le indiquemos que puede empezar. Esta indicación la realizaremos mediante la tecla ENTER.

La tecla ENTER actúa de tecla de fin de línea en el ZX-SPECTRUM.

Repetición de las teclas

El teclado del ZX-SPECTRUM está dotado de repetición automática de símbolos. Esta repetición se desencadena si mantenemos pulsada una tecla más de un segundo. Por esta razón, excepto que se diga lo contrario, debemos pulsar una tecla y soltarla inmediatamente.

Por ejemplo, si ahora pulsa la letra L y la mantiene apretada, en la pantalla le aparecerá LET LET LET... hasta llenar toda la pantalla. Hágalo y después desconecte el conector (9V DC) y conecte de nuevo. Pulse ahora la letra L una sola vez y le aparecerá en pantalla LET. A continuación pulse de nuevo la misma tecla y manténgala apretada. Verá que escribe en pantalla la letra L y la seguirá escribiendo mientras la mantenga apretada. Desconecte y conecte de nuevo.

1.2.2 Modos de funcionamimento

Después de conectar el ordenador aparece el mensaje inicial que ya conocemos. Pulsando ahora la tecla ENTER aparece el cursor con la letra K en su interior. La letra K indica que el teclado está en situación de palabra-clave («Keyword» en inglés). En general, una palabra clave es una orden que es aceptada por la máquina. Una diferencia fundamental del teclado del ZX-SPECTRUM respecto a otros es que las palabras clave se escriben con sólo pulsar una única tecla. En otros, como decimos, hay que escribir la palabra clave letra por letra. Palabras clave son, por ejemplo, LET, PRINT, INPUT, etc.

Los modos K, L y C

Siempre que aparezca una letra K dentro del cursor, entonces el ordenador está esperando una orden, es decir, está en modo palabra-clave. En esta situación, cuando se pulsa una tecla correspondiente a una letra, el ordenador escribe la palabra clave escrita en blanco sobre dicha tecla.

Por ejemplo, si pulsamos la tecla de la letra P aparece en pantalla la palabra PRINT.

Observamos ahora que en el interior del cursor no hay la letra K sino la letra L (L de LOWER CASE, minúscula en inglés). Esta letra indica que el teclado está situado en modo letra-minúscula. En esta situación, cuando se pulsa una tecla se escribe la letra minúscula correspondiente en lugar de la palabra clave como en el caso anterior. Si repetimos en este momento la operación de antes, es decir, pulsamos la tecla P, se escribe en pantalla la letra p minúscula. Por tanto, dos operaciones idénticas (la pulsación de la letra p) han dado dos resultados distintos (escribir PRINT o la letra p). Esto se debe a que hemos pulsado la tecla cuando el teclado se encontraba en dos modos distintos de operación. Pulse de nuevo la letra P y verá que aparece en la pantalla una *p* minúscula después de PRINT.

> El texto o símbolo que aparecerá en pantalla al pulsar una tecla dependerá del modo de operación del teclado. Este modo de operación viene indicado por la letra contenida dentro del cursor.

En pantalla tenemos ahora la frase:

```
PRINT p
```

La tecla ENTER

Esta frase constituye una orden para el ZX-SPECTRUM. Recordemos que hasta que no informemos al ordenador que hemos terminado de escribirla, él se quedará esperando sin tomar ninguna iniciativa. Esta orden, aunque está correctamente escrita, no tiene ningún significado en este momento. Si pulsamos la tecla de fin de línea, es decir la tecla ENTER, el ordenador recibe la orden y actúa en consecuencia. Como ya hemos dicho, la orden no tenía ningún significado, por lo que en pantalla aparece el mensaje:

2 Variable not found, 0:1

Hágalo y lo verá. Sin embargo, este mensaje no debe preocuparnos. Simplemente nos informa (desafortunadamente, en inglés) de la razón por la cual no ha obedecido la orden. Es decir, el mensaje indica que la variable no aparece.

Mensajes que da el ordenador

Siempre que demos una orden que no pueda ser obedecida, el ordenador nos dará un mensaje informándonos del problema. De esta primera experiencia con el teclado podemos extraer dos conclusiones:

a) Que el ordenador sólo actúa cuando se le informa de que hemos terminado de escribir la orden, es decir, pulsando la tecla ENTER. En caso contrario se queda esperando indefinidamente.

Si alguna vez tenemos la impresión de que el ordenador está parado y no obedece la orden, es muy probable que la causa sea que no hemos

pulsado la tecla ENTER. Le insistimos en este punto, porque al principio le pasará muchas veces.

b) Si damos una orden incorrecta, el ordenador nos da un mensaje informativo. El lenguaje BASIC se caracteriza por establecer un diálogo con el usuario. Al recibir una orden la realiza o bien informa de la razón por la cual no puede efectuarla.

1.2.3 Teclas CAMBIAR MAYUSCULAS y FIJAR MAYUSCULAS

En su ordenador, estas teclas se llaman CAPS SHIFT (CAMBIAR MAYUSCULAS) y CAPS LOCK (FIJAR MAYUSCULAS). La tecla CAPS SHIFT se encuentra en la parte inferior izquierda del teclado. Actúa de la misma manera que la tecla de cambio minúscula-mayúscula en las máquinas de escribir. Por tanto, ella sola no hace nada. En cambio, si se escribe una letra manteniendo apretada la tecla CAPS SHIFT entonces la letra se escribe en mayúsculas.

Probemos un ejemplo. Desconectemos y conectemos de nuevo el microordenador. Pulsemos primero ENTER de modo que nos aparezca el cursor con la tecla K en su interior. Tecleemos la letra P. Aparece, como ya sabemos, la palabra PRINT y el cursor contiene ahora la letra L. Si pulsamos la tecla A aparece una *a* minúscula. Si mantenemos apretada la tecla CAPS SHIFT y escribimos de nuevo una A, entonces aparece una A mayúscula. Si soltamos la tecla CAPS SHIFT y seguimos escribiendo letras, aparecen en minúsculas. (Pruebe a escribir la *s,* la *d* y la *f,* que están a continuación de la A.)

Fijación del teclado a mayúsculas

Al igual que en las máquinas de escribir, podemos fijar el teclado para que escriba siempre en mayúsculas. Si su ordenador es el ZX-SPECTRUM+, para fijar las mayúsculas basta pulsar la tecla CAPS LOCK, situada en la parte inferior del teclado. Pero si su ordenador es el ZX-SPECTRUM normal debe pulsar la tecla CAPS LOCK, que se encuentra en la tecla del número 2, manteniendo pulsada la tecla CAPS SHIFT. Hecho esto, observaremos que en el interior del cursor hay una C. Esta C indica que el teclado está en modo letra mayúscula («Capital letter» en inglés). Escriba ahora su nombre y verá que lo hace todo en mayúsculas. En pantalla tenemos ahora la palabra PRINT, seguida de varias letras, y su nombre. Veremos a continuación cómo borrar las letras.

1.2.4 Tecla BORRAR

DELETE

En su microordenador esta tecla se llama DELETE (borrar en inglés). En el ZX-SPECTRUM se encuentra en la tecla correspondiente al cero y para utilizarla, mantendremos apretada CAPS SHIFT y pulsaremos la tecla DELETE. En el ZX-SPECTRUM+ se encuentra en la parte izquierda del teclado, como tecla independiente y para utilizarla basta pulsarla, sin apretar ninguna tecla más. En consecuencia utilizaremos la tecla DELETE para borrar símbolos escritos. Al usarla, observamos que el cursor retrocede una posición y borra la última letra escrita. Atención, sin embargo, en el ZX-SPECTRUM, pues si en lugar de borrar la letra aparece un cero es que no hemos mantenido apretada suficientemente la tecla CAPS SHIFT.

Repitiendo la operación, se elimina la siguiente letra. De esta forma podemos eliminar toda la línea si hace falta o bien sólo borrar los símbolos equivocados y reescribir los símbolos correctos.

Si hay que borrar toda una línea, aprovecharemos la posibilidad de repetición automática que tiene la tecla de borrado. Procediendo como le hemos indicado para borrar, si mantenemos apretada la tecla DELETE borraremos todo lo que hay. Le aparecerá entonces la K intermitente. Para probar, escribamos varias letras sin pulsar la tecla ENTER. No importa lo que aparezca en pantalla, puesto que lo borraremos a continuación. Manteniendo apretadas las teclas de borrar veremos que la acción de borrado se repite de forma automática, mientras sigan apretadas las teclas. La repetición se detiene cuando soltamos las teclas o cuando el cursor queda situado en el inicio de línea.

En el ZX-SPECTRUM, la combinación de CAPS SHIFT y DELETE y en el ZX-SPECTRUM+ la tecla DELETE actúan de teclas de retroceso y borrado de un carácter.

1.2.5 Tecla CAMBIAR SIMBOLO

En su microordenador esta tecla se llama SYMBOL SHIFT. Hasta ahora hemos visto cómo escribir las palabras claves que están en las teclas, como PRINT, LET, etc. Pero dentro de las teclas hay otros símbolos y palabras. Por ejemplo, en la tecla A, además de la palabra clave NEW está la palabra STOP. En la letra K, además de la palabra clave LIST está el símbolo (+).

Acceso a los símbolos dentro de las teclas

Supongamos que deseamos escribir PRINT 1 + 2. Empezaremos con el cursor al inicio de línea (usaremos la tecla de borrado si hubiese algo escrito en la pantalla en este momento). Escribiremos PRINT (letra P). El cursor estará ahora en modo L o en modo C. Ambos nos sirven, puesto que sólo afectan al tipo de letra. Hay que escribir ahora el número 1 pulsando la tecla correspondiente y a continuación el símbolo (+). Este símbolo se encuentra en la tecla de la letra K. Seguidamente mantendremos apretada la tecla SYMBOL SHIFT y pulsaremos la letra K. En pantalla aparecerá el símbolo (+). A continuación escribimos el 2. En pantalla tendremos:

```
PRINT 1 + 2
```

Si no es así, usaremos la tecla DELETE para corregir los posibles errores, hasta que la instrucción esté correcta. Pulsamos ENTER y el ordenador

escribe 3 en la parte superior de la pantalla. Observe que en la parte inferior de la pantalla aparece el mensaje:

```
OK. 0:1
```

Este mensaje le indica que es correcto lo que acaba de hacer.

1.2.6 Teclas numéricas

En el ZX-SPECTRUM, las teclas numéricas están situadas en la fila superior. Para escribir los números se pulsan estas teclas sin necesidad de apretar la tecla SYMBOL SHIFT. Una advertencia importante es que no debemos confundir jamás el cero con la letra *O* mayúscula. Para distinguirlos el ZX-SPECTRUM escribe el cero con una barra cruzada (así: Ø). Si alguna vez se produce un error inesperado, es posible que sea a causa de que hemos escrito una O en lugar de un cero o viceversa. La misma advertencia se aplica para el número 1 (uno) y la letra l (ele) o la I (i mayúscula).

Símbolos gráficos y modo G

En las teclas numéricas observamos que hay también unos símbolos gráficos. Los emplearemos más adelante para elaborar dibujos en pantalla. Para utilizarlos debemos cambiar el modo de operación del teclado. En el ZX-SPECTRUM+ se cambia pulsando simplemente la tecla GRAPH. En el ZX-SPECTRUM normal se cambia manteniendo pulsada la tecla CAPS SHIFT y pulsando la tecla del número 9. Si haciendo esto pulsamos las teclas numéricas, veremos que en pantalla aparecen símbolos gráficos. Para salir del modo gráfico se pulsan las teclas del mismo modo que hicimos para entrar. Entonces el cursor retorna a modo C o al L. Hágalo y borre después la pantalla.

1.2.7 Espaciador

En el ordenador también existe la tecla equivalente al espaciador de la máquina de escribir. En el ZX-SPECTRUM+ es la tecla central alargada de la última fila del teclado. En el ZX-SPECTRUM normal se encuentra en la parte inferior derecha y lleva escrita la palabra SPACE (espacio en inglés). Al pulsar el espaciador, el cursor avanza una posición sin escribir nada. Es decir, dejando espacios en blanco.

1.2.8 Líneas con errores

Anteriormente hemos comprobado que si escribíamos una orden que no podía ser obedecida por la máquina, ésta nos informaba del problema. Sin embargo, puede ocurrir que le demos una orden mal escrita y el ordenador no la entienda. Es decir, el problema no es que no pueda obedecerla, sino que simplemente no entiende lo que le decimos. Normal-

mente, la causa es que se ha escrito algo que contiene errores ortográficos. Por ejemplo, escribamos la línea (sin pulsar ENTER).

```
PRINT HOLA"
```

Para ello borre primero todo lo que hay en pantalla, utilizando la tecla de borrado DELETE según se le indicó anteriormente. En la línea escrita vemos que nos hemos olvidado las comillas del principio. Pulsando ENTER, el ordenador recibe nuestra orden pero no la entiende. Entonces manifiesta su desconcierto escribiendo un recuadro con un interrogante (?).

Notemos la diferencia sustancial con el comportamiento en anteriores casos. En aquellos, dábamos una orden escrita correctamente pero que no podía ser llevada a término en aquel momento. Por el contrario, ahora la orden es ininteligible para el ordenador. No obstante, la solución está como siempre en la tecla DELETE. Utilizándola en la forma ya conocida, borraremos la parte errónea y reescribiremos la línea correctamente. Borre hasta que sólo quede PRINT en pantalla y escriba correctamente «HOLA». A continuación pulse la tecla ENTER.

Como hemos comprobado, el ordenador es extraordinariamente estricto respecto a los errores ortográficos o sintácticos. Incluso los programadores con gran experiencia cometen errores ortográficos, por tanto no hay que desanimarse si en estos primeros capítulos el ordenador nos da muchos errores. En cambio, sí que es importante saber cómo arreglarlos. Por esta razón no debemos seguir adelante hasta que no hayamos practicado lo suficiente con la tecla DELETE.



1.2.9 Numeración de programas

Las sentencias de un programa van numeradas. Empezando con el cursor en la parte izquierda de la línea (en modo K) escribiremos el número de la sentencia. El cursor permanece en modo K. A continuación escribimos la palabra clave y seguimos como siempre.

Por ejemplo, escribamos:

```
10 PRINT "HOLA"
```

Los comandos en el teclado

Al pulsar ENTER, esta línea desaparece de donde la estábamos escribiendo y aparece en la parte superior de la pantalla. El cursor ha vuelto al inicio de la línea otra vez en modo K para que podamos entrar otra orden. Como ya sabemos, las líneas numeradas forman parte de un programa y no se ejecutan de inmediato. Para que se ejecuten hay que escribir la orden RUN (proceso). Esta orden se encuentra sobre la tecla R. Si pulsamos las teclas RUN y ENTER se ejecuta el programa (en este caso formado por una sola línea) y se escribe en pantalla la palabra HOLA. En la parte inferior

aparece OK (conforme) indicando que el programa ha funcionado correctamente. Recordemos que el BASIC opera siempre de modo dialogante.

Los programas tienen la propiedad de que pueden ser ejecutados muchas veces ya que se conservan en memoria hasta que decidamos borrarlos. Por tanto, el programa anterior sigue estando en memoria. Si queremos verlo, pulsaremos la tecla LISTAR, que en el microordenador se denomina LIST, y se encuentra sobre la K, y luego ENTER. En pantalla aparece de nuevo el listado del programa.

Excepcionalmente, en el ZX-SPECTRUM la tecla ENTER por sí sola, también realiza la operación de LIST. Pulse la tecla ENTER y aparecerá en pantalla el programa.

```
10 PRINT "HOLA"
```

Este pequeño programa que tenemos en memoria, lo podemos probar todas las veces que queramos con la tecla RUN. Para borrarlo pulsaremos la tecla NUEVO, que en su máquina se denomina NEW (nuevo en inglés), situada sobre la A y luego ENTER. Al hacerlo, observamos que se borra la pantalla y aparece el mismo mensaje que sale al poner en marcha el ordenador.

1.2.10 Resumen de los modos de operación

En este apartado veremos un resumen de los modos de operación vistos hasta ahora. Además, describiremos sucintamente otro modo de operación (el modo E) que todavía no habíamos visto. En capítulos posteriores veremos con más detalle la utilizaciónde estos modos.

a) K: (Modo palabras-clave) Este modo de operación queda seleccionado de forma automática por el BASIC. Sólo acepta palabras clave o números de línea.

b) L: (Modo letra minúscula) En este modo de operación se usan las letras minúsculas y los dígitos.

c) C: (Modo letra mayúscula). En este modo, las letras escritas son mayúsculas. Para pasar de L a C o viceversa, en el ZX-SPECTRUM+ se pulsa la tecla CAPS LOCK y en el ZX-SPECTRUM se pulsan simultáneamente las teclas CAPS SHIFT y CAPS LOCK.

d) G: (Modo gráfico). Sólo se utiliza para escribir símbolos gráficos. Para pasar de L o C a modo G, en el ZX-SPECTRUM+ basta pulsar la tecla GRAPH y en el ZX-SPECTRUM se pulsan al mismo tiempo CAPS SHIFT y el número 9. Para salir del modo G se pulsan las teclas lo mismo que para entrar.

e) E: (Modo extendido). Sirve para acceder a los símbolos o palabras situados en la parte exterior de la tecla. Por ejemplo, en la letra Q tenemos SIN y ASN a las que se accede en modo E. Para pasar del modo L o C al modo E, en el ZX-SPECTRUM+ se pulsa la tecla EXTEND MODE, situada

en la parte izquierda del teclado y en el ZX-SPECTRUM pulsando al mismo tiempo CAPS SHIFT y SYMBOL SHIFT. Así, si estando en modo E pulsa la letra Q, en pantalla le aparecerá SIN y si estando en modo E pulsa al mismo tiempo SYMBOL SHIFT y la letra Q le aparecerá en pantalla ASN. Ahora no se preocupe de para qué sirven estos símbolos, pues ya los veremos más adelante.

1.6.2

1.2.11 Prácticas

Primera práctica

Junto este primer volumen se le ha suministrado una tarjeta para ayudarle a la localización de las teclas del ZX-SPECTRUM. La primera práctica consiste en realizar ejercicios de localización mediante la tarjeta.

Escriba las coordenadas de las teclas siguientes, tal como se indica en la primera de ellas:

1) ENTER (C, 10)
2) BEEP
3) RED
4) PAUSE
5) BLACK
6) ATN
7) GOTO
8) PRINT
9) SAVE
10) FORMAT
11) W
12) PLOT
13) COPY
14) j
15) PEEK
16) 9
17) ▞
18) %
19) COS
20) READ
21) LET
22) INPUT
23) REM
24) BORDER
25) INK
26) DRAW
27) VAL$
28) <=
29) ?
30) POKE

Segunda práctica

La segunda práctica consiste en introducir unas instrucciones en el ordenador y describir el resultado en la pantalla. Tenga en cuenta que el objetivo no consiste en entender lo que hace el ordenador sino simplemente habituarle a escribir en el teclado.

Por otra parte, compruebe que si no escribe exactamente la instrucción tal y como se la proponemos el resultado es distinto. Es muy importante que empiece a apreciar que la colaboración de todas las palabras y signos de puntuación es necesario para conseguir un efecto deseado.

Escriba ahora las sentencias siguientes:

1) CLS : LET a = 3 : PRINT «Su número de la suerte es:» ; A. Acuérdese que para que el ordenador haga algo es necesario finalizar la instrucción apretando la tecla ENTER.
   Si se equivoca utilice la tecla DELETE presionando simultáneamente la tecla CAPS SHIFT hasta retroceder al punto que está erróneo; incluso, si es necesario, hasta el inicio de la instrucción o sentencia.

2) CLS : LET A$ = «3 • 5» : PRINT VAL A$ (apriete el ENTER).

3) CLS : FOR i = 1 TO 5 : PRINT TAB (2•i);«■»; : NEXT i. Recuerde que después de escribir el símbolo gráfico, el cursor le queda en modo gráfico y debe quitarlo para continuar escribiendo la sentencia.

4) CLS : FOR i = 128 TO 143 : PRINT «!»; CHR$ i ; : NEXT i

5) CLS : FOR I = 1 TO 10 : PRINT I, SQR I : NEXT I : PRINT «12345678901234567890123456789012»

SOLUCIONES A LAS PRACTICAS

Primera práctica

1) C, 10
2) D, 2
3) A, 2
4) D, 8
5) D, 10
6) B, 3
7) C, 5
8) B, 10
9) C, 2
10) A, 10
11) B, 2
12) B, 1
13) D, 2
14) C, 7
15) B, 9
16) A, 9
17) A, 6
18) A, 5
19) B, 2
20) C, 1
21) C, 9
22) B, 8
23) B, 3
24) D, 6
25) D, 3
26) B, 2
27) C, 7
28) B, 1
29) D, 4
30) B, 9

Segunda práctica

1) Su número de la suerte es: 3. (Esta instrucción escribe simplemente un mensaje.)

2) 15. (Esta instrucción realiza el cálculo contenido en la variable textual A$.)

3) ■ ■ ■ ■ ■ (Esta instrucción escribe el símbolo gráfico de un cuadrado todo negro en las cinco primeras posiciones pares de la pantalla.)

4) (Esta instrucción escribe la lista de los 16 símbolos gráficos de que dispone el ZX-SPECTRUM, separados por un signo de admiración para diferenciarlos bien.)

5)
```
1                1
2                1 . 4 1 4 2 1 3 6
3                1 . 7 3 2 0 5 0 8
4                2
5                2 . 2 3 6 0 6 8
6                2 . 4 4 9 4 8 9 7
7                2 . 6 4 5 7 5 1 3
8                2 . 8 2 8 4 2 7 1
9                3
1 0              3 . 1 6 2 2 7 7 7
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
```

(Escribe la tabla de las raíces cuadradas de los 10 primeros números. Al final escribe una secuencia de números para identificar las columnas de la pantalla.)

Capítulo 2

ESQUEMA DE CONTENIDO

Teclado (Continuación)	Zonas de pantalla Mayúsculas y minúsculas El modo de operación extendido Nombres de variables Edición de programas
La instrucción REM	
Instrucción PRINT	El encolumnador TAB
Instrucción INPUT	Instrucción INPUT LINE
Instrucción GOTO	Detención de un programa Continuación del proceso
Prácticas con el teclado	

2.1 TECLADO (CONTINUACION)

2.1.1 Zonas de pantalla

La instrucción BORDER y la zona de presentación

El ZX-Spectrum divide la pantalla en dos zonas. Para apreciarlas con claridad, estando el cursor en modo K, pulsemos la letra B. Aparece la palabra BORDER («border» en inglés significa reborde o margen). A continuación escribamos el número 2. Al pulsar ENTER observaremos que la pantalla está formada por un recuadro central con el reborde de color rojo. De momento nos podemos olvidar de la instrucción BORDER. En el capítulo dedicado a gráficos volverá a aparecer. Para eliminar el color del reborde utilizaremos la tecla NEW y a continuación la tecla ENTER. Hágalo.

Esto nos ha permitido apreciar las dos zonas en que se divide la pantalla.

Vuelva ahora a dividir la pantalla en las dos zonas, según lo indicado.

La parte central se utiliza para la presentación de los resultados y de los listados de los programas. La parte inferior del reborde se utiliza como zona de trabajo para la entrada de órdenes o comandos, líneas de programas y para que el BASIC escriba sus mensajes informativos.

Este funcionamiento a base de zonas es típico del ZX-Spectrum, la mayoría de ordenadores operan de modo distinto, sin subdividir la pantalla. Esto hace que el cursor sólo aparezca en la zona de trabajo, nunca en la zona de presentación.

La anchura de pantalla con la que trabaja el ZX-Spectrum es muy reducida. En la zona de presentación tiene solamente 32 columnas, que a su vez están divididas en dos zonas de 16 columnas cada una, que se denominan zonas de tabulación. El funcionamiento de estas zonas las veremos en el apartado 2.3.1 de esta lección. Sin embargo, esto no significa que una línea de programa está limitada a 32 símbolos; por el contrario, puede alcanzar hasta 255. Lo único que ocurre es que al llegar al final de la pantalla, continuará por la línea inferior, pero sigue siendo una única línea de programa.

La zona de trabajo

Como podemos observar, cuando escribimos algo, utilizamos la zona de trabajo. Si se trata de una línea de programa. es decir, una sentencia numerada, cuando se pulsa ENTER, la sentencia desaparece de la zona de trabajo y queda situada en la zona de presentación. Por ejemplo, escriba para comprobarlo,

```
10 PRINT 1+8
```

Si por el contrario, le damos una orden de ejecución inmediata, por ejemplo:

```
PRINT 1+8
```

se escribe el resultado en la zona de presentación, y en la zona de trabajo aparece el mensaje

```
0 OK 0:1
```

que indica que todo ha ido bien.

Si damos la orden

```
PRINT Z
```

entonces aparece el mensaje

```
2 Variable not found 0:1
```

en la zona de trabajo, que indica que hemos cometido un error, en este caso, dar la orden de escribir una variable que no hemos definido previamente. Vamos a analizar el mensaje que ha aparecido.

Estructura de los errores

Los mensajes de error constan de tres partes. La primera de ellas es el código de error. Puede ser un número o una letra. A continuación sigue el texto del mensaje, en este caso el texto significa en inglés «variable no encontrada». Finalmente, hay un par de números separados por dos puntos (:). El primero de estos números indica en qué número de línea se ha producido el error. En este caso es el cero puesto que la orden estaba escrita sin numerar. El segundo valor indica qué instrucción es la errónea, dentro de la línea. Como sólo hemos escrito una instrucción aparece el número 1.

Recordemos que podemos escribir varias instrucciones en una sola línea separándolas por dos puntos (:). Si escribimos:

```
50 PRINT 3+4 : PRINT Z
```

Al escribir RUN y pulsar ENTER el mensaje será

```
2 Variable not found 50 : 2
```

que indica que el error se ha producido en la segunda instrucción de la línea 50.

Estas indicaciones son útiles cuando cometemos un error al escribir un programa y así se localiza fácilmente.

2.1.2 Mayúsculas y minúsculas

En el BASIC se establece que tanto las palabras clave como las variables se escribirán en mayúsculas. Sin embargo, la mayoría de variantes del BASIC aceptan también las letras minúsculas.

Respecto a las palabras clave no hay ningún problema, puesto que el ZX-Spectrum las escribe totalmente en mayúsculas al pulsar una sola tecla. Para el resto, el ZX-Spectrum utiliza en principio las letras minúsculas. Esta situación se puede cambiar, como ya sabemos, con las teclas CAPS SHIFT y CAPS LOCK. No hay diferencia alguna en utilizar un tipo de letra u otro. Tampoco hay diferencia en el nombre de una variable por el hecho de estar escrita en distinto tipo de letra. Por ejemplo:

```
LET A = 10
PRINT a
```

Las variables prescinden de si es mayúscula o minúscula

El resultado es 10, ya que hemos utilizado una única variable. Por la misma razón, los nombres de la variable AA, Aa, aA y aa corresponden a una variable única, cosa que podemos comprobar escribiendo:

```
LET aa = 15
PRINT AA, Aa, aA, aa
```

Asimismo, para la notación exponencial, se puede utilizar la letra e minúscula. Así por ejemplo, el resultado de escribir

```
LET a = 1e+6
PRINT a
```

es un millón.

El único momento en que hay que diferenciar entre letras mayúsculas y minúsculas es en los datos textuales. Pero esta diferencia sólo tiene importancia para el usuario, ya que el ordenador se limita a transcribir el dato tal cual está escrito.

2.1.3 El modo de operación extendido

Como ya sabemos, cada tecla sirve para varias cosas a la vez. Por ejemplo, en la tecla P tenemos también la palabra TAB y el signo ©. Para acceder a estas palabras o símbolos hay que pasar al modo E.

Ya sabe también que al modo E se pasa en el ZX-SPECTRUM+ pulsando la tecla EXTEND MODE y en el ZX-SPECTRUM apretando simultáneamente las teclas CAPS SHIFT y SYMBOL SHIFT. Supongamos que el

cursor está en modo K, L o C. Entonces, al pasar al modo extendido el cursor aparecerá con una E parpadeante en su interior, que indica que estamos en modo E. Si repetimos la operación, el cursor vuelve a modo K, L o C.

No olvidemos que el teclado está dotado de repetición automática. Por tanto si mantenemos apretadas las teclas que nos permiten pasar al modo E, el cursor irá pasando alternativamente del modo E al K (o L o C, según estuviera previamente) hasta que lo soltemos.

El modo extendido accede a los símbolos fuera de las teclas

Por ejemplo, escribamos PRINT y a continuación pasemos el cursor a modo E. Al pulsar la P de nuevo, aparece la palabra TAB. El modo E ha desaparecido, y estamos de nuevo en modo L (o C). Esta es una característica típica del modo E: desaparece automáticamente al utilizarlo.

En cuanto al otro símbolo que está en la misma tecla P, que se denomina «Copyright» (©), se puede acceder a ella también desde el modo extendido, pero en este caso, además de apretar la tecla cuando el cursor está en modo extendido hay que mantener apretada simultáneamente la tecla de SYMBOL SHIFT. Pruebe a escribir la instrucción.



```
PRINT " © J. GARCIA"
```

Una vez escrito el símbolo o palabra correspondiente el cursor cambia al modo L, C o G, es decir, desaparece el modo E.

2.1.4 Nombres de variables

Sin embargo, en el ZX-Spectrum existen algunas variaciones respecto a las normas estándar de nomenclatura de las variables.

Estas variaciones son:

a) Los nombres de las variables numéricas pueden contener un número indeterminado (inferior a 250) de letras o dígitos.

Ejemplos:

```
10 LET TOTAL=100
20 LET INCREMENTO = 5
30 PRINT TOTAL+INCREMENTO
```

Las variables numéricas pueden tener un nombre muy largo

En el ejemplo propuesto la variable TOTAL tiene 5 letras. La variable incremento tiene 10 letras. El hecho de que estas variables estén formadas por un conjunto de letras que forman una palabra con sentido, no debe confundirse con un texto, el cual siempre irá entre las comillas características de los textos.

b) Dentro del nombre de las variables numéricas se pueden dejar espacios en blanco.

```
40 LET TOTAL GLOBAL = TOTAL+INCREMENTO
```

La variable TOTAL GLOBAL utiliza un espacio en blanco entre la L final de TOTAL y la G inicial de GLOBAL. Aun cuando estos nombres de variables están permitidos debe evitar que las variables contengan un carácter en blanco en medio, pues cuando el programa es largo suelen fomentar las confusiones.

c) Los nombres de las variables textuales están formados por una sola letra y el símbolo dolar ($)

Las variables textuales sólo pueden tener una letra

El uso de nombres de variables con varias letras ayuda a que el programa sea mucho más legible. No obstante, si se abusa en la longitud del nombre se corre el riesgo de cometer errores al intentar repetirlo. Cuanto más largo es el nombre de una variable más fácil es equivocarse al escribirlo (cambiar una letra por otra, olvidarse de una letra, etc.).

Por otra parte, y desafortunadamente, en el caso de las variables textuales estamos muy limitados puesto que sólo podemos emplear una letra.

2.1.5 Edición de programas

Si una vez que hemos escrito un programa deseamos introducir cambios en una línea, disponemos de dos procedimientos. El primero de ellos ya nos es conocido y consiste en reescribir toda la línea incluyendo en ella las modificaciones oportunas. Este procedimiento tiene el inconveniente de que obliga a teclear de nuevo los trozos de línea que no sufren variación. Esto es especialmente molesto en el caso de líneas muy largas (con varias instrucciones) en las que sólo hay que efectuar un leve retoque.

Para evitar estos inconvenientes existe un segundo procedimiento denominado edición de una línea. En informática, la palabra *editar* tiene el significado de modificar un texto a base de borrar, reemplazar o insertar nuevos caracteres. Hay que tener presente que la pantalla ofrece muchas más posibilidades para introducir modificaciones que una máquina de escribir.

Línea en curso

Antes de aprender a modificar una línea, debemos conocer un concepto importante. Este concepto es el de *línea en curso.*

Línea en curso:

Cuando se entra un programa, existe siempre una línea que recibe el nombre de *línea en curso.* En principio, la línea en curso coincide con la última línea escrita, que no ha de ser forzosamente la última línea de programa, puesto que éstas se pueden entrar en desorden. Si escribimos el siguiente programa para convertir centímetros en pulgadas (recuerde que cada 2.54 centímetros hacen una pulgada):

```
NEW
10 INPUT "CENTIMETROS",C
20 LET P=C/2.54
30 PRINT P
```

observaremos que, a medida que entramos las líneas, aparece una marca en la línea que acabamos de escribir. Esta marca es el signo > situado entre el número y la palabra clave. Si hemos entrado las instrucciones consecutivamente, la línea 30 tendrá la siguiente apariencia:

```
30 > PRINT P
```

Este símbolo que se lee «mayor que» (>) es el que señala la línea en curso. Entremos ahora una nueva línea, por ejemplo:

```
5 REM Conversion cms. a pulgadas
```

Para desplazar la línea en curso se utiliza la flecha hacia abajo y la flecha hacia arriba

En el ZX-SPECTRUM+ tenemos en la última fila del teclado cuatro flechas: dos en sentido horizontal y dos en sentido vertical. Estas mismas flechas las tenemos en el ZX-SPECTRUM normal sobre los números 5, 6, 7 y 8. Las dos líneas verticales (flecha hacia arriba y flecha hacia abajo) se utilizan para cambiar la línea en curso. Mientras que en el ZX-SPECTRUM+ basta pulsar la tecla respectiva para utilizarlas, en el ZX-SPECTRUM se hace manteniendo apretada al mismo tiempo la tecla CAPS SHIFT.

Pulsando la tecla de flecha hacia abajo observamos que el símbolo «mayor que» (>) se desplaza hacia la línea siguiente. Pulsando la flecha hacia arriba, vemos que el símbolo «mayor que» se desplaza hacia la línea anterior. De esta forma podemos colocar el signo «mayor que» sobre cualquier línea del programa, con lo cual decidimos que sea esta precisamente la línea en curso.

Bien, ha llegado ahora el momento de editar una línea. Supongamos que queremos cambiar la línea 30, y escribirla de la siguiente manera:

```
30 PRINT P;"PULGADAS"
```

La tecla EDIT nos trae la línea en curso a la zona de trabajo

Lo primero que hacemos es colocar el signo (>) sobre la línea 30. Entonces si pulsamos la tecla EDIT aparece una copia de la línea 30 en la zona de trabajo. En el ZX-SPECTRUM+ esta tecla EDIT se encuentra en la parte izquierda del teclado y basta pulsarla para editar la línea. En el ZX-SPECTRUM normal esta tecla se encuentra en el número 1 y se debe pulsar manteniendo apretada al mismo tiempo la tecla CAPS SHIFT. El cursor aparecerá entre el número de línea y la palabra PRINT. Para moverlo utilizaremos las flechas horizontales hacia la derecha o hacia la izquierda según nos convenga.

El cursor se desplaza de una forma muy curiosa. Salta de golpe (pulsando la tecla una vez) por encima de la palabra PRINT, y se coloca entre ésta y la letra P. Al siguiente desplazamiento pasa a la derecha de la letra P y ya no es posible avanzar más. Utilizando ahora la flecha hacia la izquierda, podremos hacer retroceder el cursor hacia el inicio de la línea. Vemos pues, que podemos mover el cursor por encima de la línea hacia delante y hacia atrás y situarlo en la posición que deseemos.

Las flechas hacia la izquierda y hacia la derecha nos mueven por la instrucción

En el ejemplo que estamos viendo, queremos añadir algo al final de la línea. Por tanto, situaremos el cursor en este punto y escribiremos entonces el punto y coma (;) y la palabra PULGADAS entre comillas (''). Si movemos ahora el cursor, veremos que puede desplazarse sobre toda la línea, incluyendo el texto que acabamos de introducir. Una vez finalizada la edición, pulsamos ENTER y entonces la línea 30 desaparece de la zona de trabajo y queda situada en el programa sustituyendo a la antigua.

La línea en curso sigue siendo la 30. Por tanto, si pulsamos EDIT de nuevo en la zona de trabajo aparece la línea 30. Pulsando ENTER terminaremos la edición de la línea.

Desplacemos ahora el signo «mayor que» (>) hasta la línea 10 y, a continuación, pulsemos EDIT. En la zona de trabajo aparece la línea 10. Desplacemos horizontalmente el cursor hacia el final de la palabra CENTIMETROS justo antes de las comillas (''). Escribamos ahora, por ejemplo el signo igual (=). La parte derecha de la línea se ha desplazado, y el signo igual (=) ha quedado insertado. Si pulsamos la tecla DELETE, se borra el signo igual (=) y la parte derecha de la línea se ha desplazado. Esta es la manera que emplearemos para insertar o borrar caracteres. Para reemplazar un carácter por otro, se borra el antiguo y se escribe el nuevo. Pulsando ENTER damos por finalizada la edición de la línea 10.

A veces, el signo «mayor que» (>) no aparece en el listado del programa. Escribiendo un 25 y pulsando ENTER, el signo (>) desaparece. Esto se debe a que la última línea con la que hemos trabajado es la número 25. Lo que ha sucedido es que hemos dejado esta línea en blanco, y por tanto no existe como tal en el programa. El indicador de línea en curso (>) ha quedado situado entre las líneas 20 y 30 y por tanto es invisible. Sin embargo, si pulsamos las teclas de movimiento en vertical, por ejemplo la flecha hacia abajo, el símbolo aparece de nuevo, sobre la línea 30.

Ahora ejecute el programa y averigüe cuántas pulgadas son 62 cm.

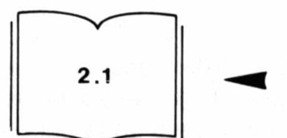
Mediante la tecla DELETE se puede suprimir cualquier parte de la instrucción

Si realizamos la edición de una línea, por ejemplo la 30, veremos que podemos cambiar incluso el número de línea. Por ejemplo, cambiemos el 30 por un 40. Para ello, deberemos borrar el 3 con la tecla DELETE y escribir un 4 a continuación. Al pulsar ahora la tecla ENTER, veremos que aparece una nueva línea en el programa, la 40, además de la 30 de la cual habíamos partido. Este sistema lo emplearemos para dos cosas: la primera para realizar copias de líneas cuando haya que escribir varias similares. Por ejemplo, en el programa:

```
NEW
10 PRINT "**********"
20 PRINT "   HOLA"
30 PRINT "**********"
```

La línea 30 se puede obtener editando la número 10 y cambiando su número de línea.

El segundo caso para el que utilizaremos el cambio de número será para trasladar una línea dentro del programa de una posición a otra. Al cambiar el número de línea obtenemos una copia (con alguna modificación si se desea) de la línea inicial que quedará situada en el lugar que le marque el número de línea. Si borramos entonces la línea inicial (escribiendo el número de la misma y pulsando ENTER), el efecto global será el de trasladar una línea dentro del programa.



2.2 LA INSTRUCCION REM

Su microordenador ZX-Spectrum posee la instrucción REM y funciona de acuerdo con las líneas generales del BASIC estándar.

Sólo es necesario hacer una observación cuando se quieren escribir sentencias que contienen un REM y sirven para eliminar una instrucción.

Suponga que desea escribir las sentencias siguientes:

```
10 REM INPUT "SU NOMBRE, POR FAVOR: ",N$
30 REM PRINT N$
```

Si intenta escribirla con su ZX-Spectrum no podrá escribir seguidos el REM y el INPUT, y tampoco el REM y el PRINT.

Este comportamiento se debe a que una vez escrita la instrucción REM, el cursor le queda en modo de operación L o C, y, por lo tanto, la instrucción INPUT y la instrucción PRINT son inaccesibles en este modo del cursor.

Escríbalas y lo máximo que podrá hacer es una intervención como la siguiente:

```
10 REM i "SU NOMBRE, POR FAVOR: ",N$
30 REM p N$
```

en efecto, cuando quiere escribir INPUT, pulsa la tecla que contiene la i y el INPUT, pero como está en el modo de operación L, se le presenta la i. Exactamente el mismo razonamiento se puede aplicar la sentencia numerada con el 30, en donde le saldrá una p por pulsar la tecla PRINT con el modo operación L. Escriba las dos sentencias y compruebe el resultado.

Por otra parte, puede mejorar la representación escribiendo INPUT y PRINT con todas las letras. Hágalo y compruebe que puede escribir la sentencia del modo siguiente:

```
10 REM INPUT "SU NOMBRE, POR FAVOR: ",N$
```

o

```
30 REM PRINT N$
```

Recuerde que tanto el INPUT como el PRINT escritos de esta manera son instrucciones falsas. Para comprobarlo, utilice el editor de programas y edite la sentencia 10. Mediante el cursor sitúese en la posición inmediatamente después de REM y borre, con la tecla DELETE, el REM. Dele a continuación la tecla ENTER y observará que le aparece un interrogante en la posición inmediatamente después del 30 e inmediatamente antes del INPUT. La razón es que, como sabe, el ZX-Spectrum requiere que las instrucciones se escriban con una sola pulsación de tecla.

Para salir de esta situación vuelva a colocar el REM en su sitio.

Haga lo mismo con la instrucción 30 y verá que el efecto es exactamente el mismo.

Finalmente queremos señalarle que hay dos soluciones más prácticas cuando lo que queremos es anular una sentencia pero manteniendo su forma original para poderla recuperar con facilidad.

Colocación del REM en una sentencia que se quiere anular

La primera consiste en que después del REM coloque los dos puntos de separación de instrucciones en una sentencia. Pruebe.

Ahora lo que debe quedarle escrito es:

```
10 REM : INPUT "SU NOMBRE, POR FAVOR: ",N$
```

y

```
30 REM : PRINT N$
```

El efecto de los dos puntos es devolverle el cursor en el modo de operación K, de tal manera que pueda escribir el INPUT y el PRINT como palabras clave.

Observe que a la hora de la ejecución, ninguna de las instrucciones que componen la sentencia con el REM al inicio de la sentencia se ejecutará y, por lo tanto, si quiere recuperar la sentencia para que se ejecute, sólo le bastará con eliminar la instrucción REM y los dos puntos que siguen a continuación.

La segunda solución consiste en utilizar el editor. Entre en primer lugar las instrucciones con INPUT y PRINT correctamente.

Una vez entradas y utilizando el editor de programas, llámelas para modificar. El cursor le queda situado en modo K y en la posición inmediatamente detrás del número de sentencia; pulse en estos momentos la tecla REM, el cursor está en modo K y, por lo tanto le intercalará en el texto la palabra REM.



2.3 INSTRUCCION PRINT

En el ZX-Spectrum el cursor nunca aparece en la zona de presentación de los datos. Sin embargo, la instrucción PRINT realiza el salto de línea, aunque este efecto no sea visible hasta que se ejecute otra instrucción PRINT. Para impedir el salto de la línea se escribe un punto y coma (;) al final de la instrucción.

Ancho de pantalla 30 caracteres

La zona de presentación tiene una anchura de 32 caracteres (de 1 a 32) y está dividida en zonas de tabulación. La primera empieza en la columna 1 y la segunda en la 17.

Dos zonas de tabulación de 16 caracteres

En otros modelos de ordenadores con pantallas de hasta 80 columnas, existen más zonas de tabulación. En el ZX-Spectrum sólo existen dos zonas. Por tanto, si se imprimen más de dos resultados, al superar la segunda zona, se salta de línea y se empieza de nuevo por la izquierda.

En el programa:

```
10 INPUT N$
20 PRINT "12345678901234567890123456789012"
30 PRINT "HOLA...",N$," QUE HAY ?"
```

La línea 20 de este programa describe 32 números que permiten visualizar cada una de las columnas para localizar fácilmente la situación del texto. Esta línea aparecerá frecuentemente en los ejemplos siguientes, recuerde que es una facilidad para la localización de las columnas.

Si contestamos PEPE en la pregunta 10, el resultado en pantalla será:

```
12345678901234567890123456789012
HOLA...         PEPE
 QUE HAY ?
```

La palabra HOLA empieza en la columna 1 y la palabra PEPE en la 17. La frase QUE HAY? empieza en la 1, pero debe tenerse en cuenta el espacio en blanco que hemos escrito antes de la Q. Por tanto, la frase se escribe a partir de la columna 2.

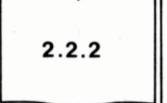
Los resultados numéricos no llevan espacios en blanco

Una diferencia importante del BASIC del ZX-Spectrum respecto al BASIC estándar es que los resultados numéricos no llevan espacios en blanco al inicio y al final del número. Por tanto, en el ZX Spectrum, la instrucción

```
PRINT 8;"METROS"
```

dará como resultado 8 METROS.

Conviene tener presente esta particularidad cuando intentemos adaptar un programa escrito para este ordenador, ya que muchos programadores aprovechan los espacios en blanco que deja el BASIC estándar de forma automática, en los resultados sucesivos.



2.3.1 El encolumnador TAB

A la tecla TAB se accede mediante el modo de operación E. En la tecla que contiene la P.

Escribamos la instrucción.

```
PRINT TAB(15);"HOLA"
```

Esta instrucción escribirá la palabra HOLA en la columna 16.

Es decir, el encolumnador TAB deja tantos espacios (que puede estar en blanco o no) desde el inicio de la línea como indica el número que va en el interior del paréntesis.

En el ZX-Spectrum no es imprescindible la colocación de paréntesis. La siguiente instrucción también sería válida:

```
PRINT TAB 15 ;"HOLA"
```

No obstante, se recomienda no omitir los paréntesis, pues los programas dejan de ser compatibles con el BASIC estándar en el que es obligatorio el uso del paréntesis.

El ZX Spectrum tiene una particularidad en la utilización del TAB, que difiere del BASIC estándar. En el BASIC estándar, si ya se ha escrito después de una columna no se puede utilizar el TAB con un número inferior a ella.

Por ejemplo, si se escribe el programa:

```
10 PRINT "12345678901234567890123456789012"
20 PRINT "ESTO ES UN ";TAB(5);"ERROR"
```

el BASIC estándar no lo admite, pues no tiene sentido escribir ERROR en la quinta columna, cuando la N de UN ya ocupa la columna 10. El ZX-Spectrum, sin embargo, aunque no lo puede escribir lógicamente en la co-

lumna 6 de la misma línea, pues ya está ocupada, lo escribe en la columna 6 de la siguiente línea de pantalla que está libre. Ejecute el programa anterior y verá con claridad el comportamiento.

La instrucción TAB no retrocede pero cambia de línea

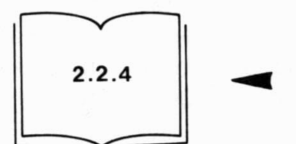
Si dentro de la función TAB se emplea un número inferior a 31, entonces el comportamiento es normal. Si el número es mayor que 31, se realiza automáticamente la división del número que hemos colocado en el interior del TAB y se divide por 32, tomando como nuevo número el resto de esta división.

Por lo tanto, la instrucción

```
PRINT TAB(40);"HOLA"
```

imprimirá la palabra HOLA en la columna 9, dado que 40 dividido por 32 es igual a 1 y que da de resto 8, lo que equivale a haber colocado en el encolumnador TAB la cifra de 8.

En líneas generales Ud. puede colocar en el interior de los paréntesis del TAB el número que quiera, pero automáticamente el ZX-Spectrum le cambiará (a la hora de ejecutar la instrucción) este número por el resto de dividirlo por 32.



2.4 INSTRUCCION INPUT

La instrucción INPUT tiene algunas particularidades respecto al BASIC estándar.

El INPUT se coloca siempre en la zona de trabajo

— En primer lugar, cuando esta instrucción pregunta un valor, lo hace siempre en la zona de trabajo. Una vez se ha pulsado ENTER, esta zona queda borrada.

— La segunda particularidad es que no emplea el signo de interrogación (?) cuando pide un dato. El BASIC estándar coloca siempre el signo de interrogación (?) cuando ejecuta una instrucción INPUT. En el ZX-Spectrum esto no es necesario ya que divide la pantalla en dos zonas. Entonces, si durante la ejecución de un programa el cursor aparece en modo L en la zona de trabajo, significa que se está ejecutando una instrucción INPUT, la cual nos pide un dato.

Por otra parte, si la variable es de tipo textual, al pedir la contestación en la zona de trabajo el propio ordenador coloca las comillas (") entre las cuales escribiremos el texto. En el ZX-Spectrum, las comillas son siempre obligatorias para la entrada de datos textuales.

— La tercera particularidad, se manifiesta cuando se entran más de una variable. Cuando se ejecuta el programa:

```
10 PRINT "12345678901234567890123456789012"
20 INPUT N$,A$,B$
30 PRINT N$,A$,B$
```

el cursor se colocará entre comillas (N$ es textual) en la columna 1 de la zona de trabajo. Cuando pulsemos ENTER, se colocará en la columna 17 para pedir el valor de A$. Finalmente, al pulsar ENTER de nuevo, se desplazará a la línea inferior, y en la columna 0 de esta línea, esperará que introduzcamos el valor de B$. Este comportamiento es totalmente distinto del BASIC estándar, donde se hubieran debido contestar los tres valores seguidos, separados por comas, pulsando ENTER una sola vez, después de haber escrito los tres. La ventaja del ZX-Spectrum en este caso, es que no hay posibilidad de contestar más o menos valores que los preguntados. En efecto, los datos se preguntan uno detrás de otro, y no es posible entrar más de la cuenta puesto que se borra la zona de trabajo, después de entrar en el último dato.

Al entrar varias variables se pueden utilizar la coma y el punto y coma

Este hecho, unido a que los datos textuales van obligatoriamente entre comillas (''), evitan los problemas de la existencia de comas (,) formando parte del texto.

— La última particularidad del ZX-Spectrum consiste en el empleo de separadores entre el texto informativo y las variables. En el ZX-Spectrum, los símbolos que utilizamos para separar (separadores) y que son la coma (,) y el punto y coma (;), se comportan como en una instrucción PRINT. Si cambiamos la anterior línea 20 por

```
10 PRINT "12345678901234567890123456789012"
20 INPUT N$;A$;B$
30 PRINT N$;A$;B$
```

entonces el BASIC pedirá cada dato justo al lado de donde hemos terminado de escribir el anterior, en lugar de situarse en las sucesivas zonas de tabulación.

Esto también es válido si se emplea un texto informativo. Por ejemplo:

```
10 INPUT "NOMBRE",N$
```

colocará el texto en la columna 1 de la zona de trabajo, situándose el cursor en la columna 17, para que introduzcamos el valor de N$. Por el contrario, la instrucción

```
10 INPUT "NOMBRE";N$
```

colocará el cursor justo al lado de la palabra NOMBRE.

2.4.1 Instrucción INPUT LINE

Esta instrucción es parecida a la instrucción INPUT. La palabra LINE se encuentra en modo E en la tecla que corresponde al número 3. La instrucción INPUT LINE significa «entrada de una línea». Para escribirla debe pulsar primero INPUT y a continuación LINE. Sólo se puede especificar la entrada de una variable textual. Por ejemplo,

```
10 INPUT LINE A$
20 PRINT A$
```

Cuando se ejecuta, no aparecen las comillas, aunque el ordenador se comporta como si estuvieran.

Ejecute el programa anterior y verá que en la zona de trabajo aparece el cursor en modo L, sin las comillas típicas de la instrucción INPUT. Introduzca unos cuantos espacios en blanco y a continuación JUAN y apriete la tecla ENTER.

Verá cómo al escribirlo en la zona de presentación el resultado será el mismo que si utilizáramos el INPUT. Unicamente ha cambiado la presentación de la zona de trabajo.

Esta instrucción puede tener interés si el programa que escribimos va a ser utilizado por alguien que no está familiarizado con la informática, y encontraría extraño el uso de comillas.

2.3

2.5 INSTRUCCION GOTO

El BASIC acepta las dos formas de escribir esta instrucción, es decir GOTO, y GO TO. En el ZX-Spectrum se escribe separado de forma automática. La tecla GOTO se encuentra sobre la tecla correspondiente a la letra G.

Si escribimos el programa:

```
10 PRINT "HOLA"
20 PRINT "ADIOS"
30 GOTO 10
```

La palabra Scroll? aparece cuando se ha escrito una pantalla

al llegar al final de la pantalla, aparece en la parte inferior de la misma la palabra *scroll?*

Cuando este mensaje aparece, significa que se ha escrito una pantalla completa, con lo que el ordenador espera que hayamos tenido tiempo de leer lo que ha escrito, antes de seguir escribiendo. Si pulsamos ENTER, el ordenador sigue escribiendo, desplazándose el texto hacia la parte superior de la pantalla por donde va desapareciendo, a medida que se escribe el nuevo texto por la parte inferior. («scroll» significa en inglés desplazamiento

hacia arriba.) Cuando se completan 22 líneas, aparecerá de nuevo la palabra «scroll ?».

2.5.1 Detención de un programa

Cuando un programa cae en un proceso sin fin, hay que detenerlo utilizando la tecla de interrupción. En el ZX-Spectrum se pueden dar tres casos:

a) El programa escribe en pantalla.

Se puede detener en el scroll

Es el caso que acabamos de ver. Si cuando aparece el mensaje «scroll ?» se pulsa una *n* (de NO) o la tecla SPACE, el programa se detiene, apareciendo el mensaje:

```
D-Break- CONT repeats
```

La traducción literal de este mensaje es un tanto difícil de entender. Sin embargo, nos informa que, a pesar de que hemos contestado que no, si nos hubiéramos equivocado, es posible reemprender la ejecución mediante la tecla de continuación como se explica en el apartado 2.5.2 de estas Prácticas.

Si por el contrario a la pregunta de «scroll ?» se contesta pulsando cualquier otra tecla, el programa prosigue su ejecución.

b) El programa no escribe.

La tecla BREAK detiene el programa cuando calcula

Si el programa queda en un bucle sin salida dentro del cual no hay ninguna instrucción PRINT, entonces hay que detenerlo con la tecla de interrupción. Esta es la tecla BREAK (significa rotura, interrupción) y en el ZX-SPECTRUM+ se encuentra en la parte derecha de la primera fila y basta pulsarla. En el ZX-SPECTRUM se encuentra sobre la tecla SPACE y para utilizarla se mantiene pulsada la tecla CAPS SHIFT. Vamos a ver un ejemplo de este caso. Para ello escribiremos el siguiente programa:

```
NEW
10 LET A = 2*5
20 GOTO 10
```

Escribiendo RUN, el programa entra en un proceso sin salida, que no incluye ninguna escritura. Pulsando BREAK el programa se interrumpe. Esta tecla se puede utilizar en cualquier caso, también si el programa incluye alguna instrucción de escritura, para interrumpirlo sin necesidad de esperar a que aparezca la palabra «scroll?».

c) Entrada de datos

La introducción de STOP detiene el programa en la entrada

Cuando hay una entrada de datos (como consecuencia de una instrucción INPUT), dentro de un bucle, puede ser muy difícil disponer del tiempo suficiente para pulsar la tecla de interrupción BREAK. Por ejemplo, sea el programa:

```
NEW
10 INPUT "NUMERO",N
20 LET A = 2*N
30 GOTO 10
```

Al escribir RUN, el programa preguntará un número. Cuando hayamos dado la respuesta, el BASIC sólo necesitará una fracción de segundo para realizar el cálculo 2*N, volviendo inmediatamente a la sentencia número 10, donde volverá a realizar la pregunta. Tenga en cuenta que en pantalla no aparece el cálculo porque no le hemos dado la orden de escribir. En este caso, aunque pulsemos la tecla de interrupción, observaremos que ésta no funciona dentro de un INPUT. Por otra parte, no disponemos de tiempo material para pulsarla entre las líneas 20 y 30. Por tanto, no podemos detener el proceso con la tecla de interrupción. ¿Qué podemos hacer en este caso, para detener la ejecución del programa?

En este caso, pulsaremos la tecla de STOP, que se encuentra sobre la tecla correspondiente a la letra A. Pulsando ENTER a continuación, el programa se detiene, dando el siguiente mensaje:

```
H STOP in INPUT, 10:1
```

que significa STOP en los datos de entrada.

Si estamos entrando una variable textual el ZX-Spectrum, para poder entrar el STOP debe eliminar las comillas mediante las teclas de avance de cursor y la tecla de borrado.



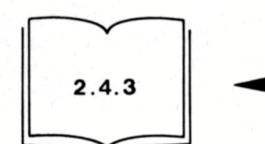
2.5.2 Continuación del proceso

La tecla de continuación, que lleva escrita la palabra CONT, se encuentra sobre la C. Escribamos el programa:

```
NEW
10 LET I = 0
20 LET I = I+1
30 PRINT I
40 GOTO 20
```

Cuando se quiere continuar no se debe alterar el programa

Cuando lo ejecutemos, veremos que van apareciendo en pantalla números consecutivos hasta que lo detenemos utilizando la tecla BREAK. Para reanudar el proceso, pulsaremos la tecla CONT. Veremos en pantalla la palabra CONTINUE. Al pulsar ENTER veremos como el programa continúa el proceso interrumpido. Va escribiendo en pantalla números correlativos, empezando por el consiguiente al último escrito antes de la interrupción.



2.6 PRACTICAS PARA EL TECLADO

Con el fin de que se familiarice con todos los símbolos de su ZX-SPECTRUM creemos que nada mejor que entablar un diálogo con su ordenador y que sea él mismo quien le vaya indicando las teclas que usted debe pulsar. Por eso, la práctica consiste en entrar al ordenador este programa que le ofrecemos y al ejecutarlo ir pulsando las teclas que el ordenador le vaya pidiendo. (Vea el programa en el apartado 2.6.6).

Pero antes de empezar queremos hacerle estas advertencias:

— *Primera:* Usted *no va a entender* lo que hace este programa y por qué lo hace. Estamos en la unidad didáctica 2 y muchas de las instrucciones que usted va a introducir no las ha estudiado aún. Sin embargo, al tener que entrar el programa es ya un buen ejercicio de localización de teclas y es de lo que se trata en este momento.

— *Segunda:* Para que el programa funcione correctamente es necesario que el texto que usted entre en el ordenador sea *exactamente igual* al que le proponemos. Cualquier símbolo incorrectamente colocado o interpretado puede impedir el correcto funcionamiento del programa. ¡Mucha atención!

— *Tercera:* Al introducir este programa usted se va a equivocar más de una vez. Los programadores profesionales también se equivocan y yo me he equivocado al escribirlo. Sería realmente una excepción que usted no se equivocara. Por tanto, cuente con ello.

— *Cuarta:* Le recomendamos hacer esta práctica en un momento en que no tenga que estar pendiente del reloj. El introducir el programa le llevará un buen rato y una vez que lo tenga correctamente en el ordenador, merece la pena que pueda practicar con tranquilidad y aprovechar así el esfuerzo realizado.

2.6.1 Guía para la introducción del programa

En primer lugar conecte el ordenador, si no lo tiene conectado y, cuando aparezca el mensaje:

© Sinclair Research Ltd.,

está todo listo para comenzar.

Le recomendamos que tenga la tarjeta de «Localización de los símbolos» a mano, pues tendrá que introducir comandos que todavía no ha estudiado. Para facilitar el trabajo, le daremos algunas indicaciones para algunas líneas en concreto. El listado del programa lo tiene al final de esta práctica.

Línea 10

Para teclear RANDOMIZE debe pulsar la tecla (B,5), en la que aparece escrita la abreviatura RAND. Para teclear CLS, pulse (la tecla D,5). Cuando haya terminado de escribir esta primera línea, no se olvide de pulsar la tecla ENTER de fin de línea. Tanto en esta línea como en todas las del programa, procure asegurarse de que está correctamente escrita antes de pulsar el ENTER.

Línea 20

La instrucción PRINT la ha escrito ya muchas veces. Por tanto, sabe dónde localizarla. Para escribir AT, pulse la tecla SYMBOL SHIFT y al mismo tiempo la tecla AT (B,8).

Línea 70

Para escribir GO SUB, pulse la tecla (C,6).

Línea 80

Después de pulsar REM (B,3), complete la línea con subrayado. El signo subrayado se encuentra en rojo sobre la tecla del cero (A,10). Le recordamos que para acceder a estos signos hay que pulsar al mismo tiempo la tecla SYMBOL SHIFT. Esto mismo tendrá que hacerlo en las líneas 2000 y 9000.

Línea 100

Para escribir IF, pulse la tecla (B,7). Para escribir THEN, la tecla (C,5) y al mismo tiempo la tecla de SYMBOL SHIFT. Para escribir GO TO, la tecla (C,5).

Línea 130

Deje 15 espacios en blanco entre las comillas. Estos mismos espacios debe dejarlos antes de cerrar las comillas en la línea 160 y 2020.

Línea 140

Para escribir BEEP ponga primero el Cursor en modo E. A continuación pulse la tecla BEEP (D,2) y al mismo tiempo la tecla SYMBOL SHIFT.

Línea 150

Para escribir FLASH pulse la tecla (D,5). Debe proceder lo mismo que en la línea 140.

Línea 170

Deje 25 espacios en blanco entre las comillas.

Línea 2030

Para escribir RETURN, pulse la tecla (B,6).

Línea 2500

Para escribir CHR$ ponga el cursor en modo E y a continuación pulse la tecla (B,7).

Línea 2570

Para escribir RND ponga el cursor en modo E y a continuación pulse la tecla correspondiente (B,5).

Línea 2580

Para escribir AND pulse la tecla SYMBOL SHIFT y al mismo tiempo la tecla AND (B,6). Para escribir OR, pulse la tecla SYMBOL SHIFT y al mismo tiempo la tecla OR (B,7). Ojo a las comillas en esta misma línea, que van escritas cuatro veces.

Línea 9030

Para escribir PAUSE, pulse la tecla (D,8).

A medida que entra las instrucciones, éstas se van colocando en la parte superior de la pantalla. Como no caben todas en una pantalla, verá que van desapareciendo. No se preocupe, porque están dentro del programa, aunque no se puedan visualizar todas al mismo tiempo.

2.6.2 Listado completo y repaso

Ya le hemos indicado que para que el programa funcione debe estar escrito todo correctamente y ser idéntico al que le proponemos. Por eso, vamos a emplear unos minutos en repasarlo.

Para ello, pulse la tecla LIST (C,8) y a continuación ENTER. Verá que en pantalla aparecen las líneas 10 y siguientes hasta llenar la pantalla. En este momento el ordenador se parará con el mensaje, en la parte inferior de la pantalla de *scroll.* Repase todo lo que tiene escrito en pantalla, letra por letra y signo por signo. En las líneas que descubra algún error o dife-

rencia con el programa propuesto, anote su número para corregirla después.

Cuando haya corregido lo que tiene en pantalla, pulse cualquier tecla y el ordenador continuará listando el programa. De nuevo aparecerá el mensaje de *scroll*. Repase ahora esta parte, haciendo lo mismo. Cuando haya acabado, pulse otra vez cualquier tecla, y le aparecerá el resto del programa. Repáselo también.

Cuando haya acabado, puede listar de nuevo el programa si lo desea, apretando la tecla LIST (C,8) y a continuación ENTER. Puede practicar el listado las veces que desee.

2.6.3 Modificación de las líneas equivocadas

En los tipos de errores que hemos podido cometer al escribir el programa, vamos a considerar dos casos:

— El primero, es que el error esté después del número de línea. En este caso basta escribir la línea completa asegurándose de que está correcta y al pulsar la tecla ENTER, la nueva línea sustituirá a la que tenía con error.

— El segundo caso, es cuando el error está en el número de línea. Es decir, que ha escrito un número distinto del que le corresponde. En esta situación, teclee primero el número de línea que usted escribió incorrectamente y a continuación la tecla ENTER. De esta manera desaparecerá toda esta línea. A continuación escriba la línea correcta otra vez, cuidando de que el número de línea también es correcto. Al pulsar la tecla ENTER, esta línea se insertará en el lugar que le corresponde.

Evidentemente, también puede hacer las correcciones editando la línea que contiene error y borrar la parte incorrecta, escribiéndola después correctamente.

Cuando ya esté seguro de que no hay errores, pase a la ejecución del programa.

2.6.4 Ejecución del programa

Ahora que estamos seguros de que el programa es correcto, vamos a ejecutarlo. Para ello, pulse la tecla RUN y a continuación la tecla ENTER.

En este momento aparecerá en pantalla:

```
PULSE LA TECLA ......:O

PULSE LA TECLA PEDIDA Y
A CONTINUACION, ENTER

PARA ACABAR PULSE SOLO ENTER

TECLA: " L "
```

Como puede ver, en la pantalla aparece la indicación de la tecla que usted debe pulsar, que en este caso es la O mayúscula.

En la parte inferior de la pantalla se le indica lo que debe hacer y cómo acabar el programa, cuando ya no deseemos practicar más.

Finalmente, en la última línea de la pantalla aparece el mensaje: TECLA: «L», con el cursor parpadeante situado encima de la L. El cursor nos indica dónde aparecerá la tecla cuando la pulsemos.

Antes de seguir adelante, y antes de haber pulsado la O mayúscula, acabemos el programa pulsando ENTER. Cómo no hemos apretado ninguna tecla, el programa acabará con el mensaje:

HA TENIDO O ACIERTOS SOBRE
UN TOTAL DE O TECLAS.

El ordenador esperará unos 20 segundos y finalmente aparecerá el mensaje:

0 OK, 9030:1

Ahora pulse de nuevo la tecla RUN y a continuación ENTER. De nuevo le pedirá que pulse la tecla O mayúscula. Hágalo.

Al pulsar ENTER, el ordenador le dice MUY BIEN! y emite un sonido.

A continuación le pedirá que pulse la tecla *s* minúscula. Deliberadamente vamos a equivocarnos y pulsamos otra tecla. Por ejemplo, la tecla *a*. Ahora el ordenador emitirá un sonido distinto y en pantalla aparecerá en la segunda fila:

HA PULSADO LA TECLA: a

y debajo el mensaje que dice: NO! VUELVA A INTENTAR, de modo parpadeante.

Acabaremos estas pruebas pulsando sólo ENTER.

Si observa que el programa no funciona como le hemos indicado, es que se ha escapado algún error al entrar el programa. Por eso, lístelo de nuevo y corríjalo.

2.6.5 Seguimiento del funcionamiento del programa

Llegado este momento, vamos a comenzar a practicar. Pulsemos de nuevo RUN y comenzamos:

El ordenador nos pide que pulsemos la O mayúscula. Si lo hemos hecho correctamente, nos pide que pulsemos la letra *s* minúscula. En caso contrario, nos avisará del error y nos dirá que lo intentemos de nuevo.

El tercer símbolo que preguntará el ordenador será una tecla de gráficos. Para ello debemos entrar en modo gráfico (repase el apartado 1.2.6 de Prácticas, si no recuerda cómo se hace) y el cursor pasará a modo G. Consulte ahora la tarjeta para localizar este gráfico. Consiste en apretar la tecla (A,3), pero en mayúscula. No se olvide de salir de modo gráfico antes de continuar.

El cuarto símbolo pedido es la tecla EXP. Esta tecla se encuentra en modo E. Por tanto debe pasar a él y pulsar la tecla pedida.

La siguiente tecla seleccionada será la ATN. También requiere el modo E y a continuación pulsar SYMBOL SHIFT y la tecla pedida.

La sexta tecla pedida será el signo de potenciación o exponencial ↑. Bastará apretar el símbolo pedido al mismo tiempo que la tecla SYMBOL SHIFT.

A partir de este momento, la selección de teclas es imprevisible y el ordenador irá pidiendo teclas hasta que usted decida dar por finalizada la sesión. Para acabar pulse sólo ENTER. Le recomendamos que haga sesiones de al menos 50 teclas hasta que el teclado le resulte totalmente familiar.

Antes de acabar esta práctica queremos hacer unas cuantas advertencias:

— Cuando le pida un signo gráfico, consulte la tarjeta si el que le pide no es exactamente el mismo que usted tiene dibujado en las teclas de los números. Como referencia le diremos que cuando le pida el inverso del que tiene usted sobre la tecla, además del modo gráfico, debe pulsar la tecla correspondiente en mayúscula. Por ejemplo, sobre la tecla (A,8) tiene usted este símbolo □. Si el que pide es este otro ■, debe pulsar la misma tecla pero en mayúscula.

— Cuando haya terminado de escribir correctamente un símbolo gráfico, no se olvide de salir del modo G. Este error es muy frecuente, pues al comienzo se olvida. El ordenador le dará error, aunque usted haya pulsado la tecla correcta. Mucha atención a este punto.

— Hay dos signos que en pantalla salen un poco diferentes a como los tiene usted dibujados en el teclado. Estos signos son la tilde, ˜ (C,1) que en la pantalla salen como unas comillas desdibujadas y el símbolo de arroba, @ (A,2), que en la pantalla le sale como una *a* al revés. Algo así (ɐ).

— El ordenador nunca selecciona los símbolos a los que se accede en modo K, por ejemplo PRINT, INPUT, etc., pues son inaccesibles desde el propio BASIC. Pero usted ya los ha practicado al entrar en el programa.

Ahora sólo nos queda felicitarle por haber realizado la práctica.

2.6.6 Listado del programa

```
  10 RANDOMIZE : CLS
  20 PRINT AT 1,1;"PULSE LA TECLA.....:"
  30 PRINT AT 17,1;"PULSE LA TECLA PEDIDA Y"
  40 PRINT AT 18,1;"A CONTINUACION, ENTER"
  50 PRINT AT 20,1;"PARA ACABAR PULSE SOLO
     ENTER"
  60 LET N = 0 : LET C = 0 : LET F$=""
  70 GO SUB 2000
  80 REM-------------------------------------
  90 INPUT " TECLA:";A$
 100 IF A$ = C$ THEN GO TO 160
 110 IF A$ = F$ THEN GO TO 9000
 120 PRINT AT 3,1;"HA PULSADO LA TECLA:"
 130 PRINT AT 3,21;A$;"          "
 140 BEEP .5,-2 : BEEP 0.25,0
 150 PRINT AT 5,1;FLASH 1; "NO! VUELVA A
     INTENTAR"; FLASH 0 : GO TO 200
 160 PRINT AT 5,1;"MUY BIEN!           ";
 170 PRINT AT 3,1;"                         "
 180 BEEP .25,6 : BEEP .25,8
 190 LET C = C + 1 : GOSUB 2000
 200 LET N = N + 1
 210 GO TO 90
2000 REM-------------------------------------
2010 GO SUB 2500
2020 PRINT AT 1,21;C$;"          ";
2030 RETURN
2500 IF C = 0 THEN LET C$ = CHR$ 79
2510 IF C = 1 THEN LET C$ = CHR$ 115
2520 IF C = 2 THEN LET C$ = CHR$ 140
2530 IF C = 3 THEN LET C$ = CHR$ 185
2540 IF C = 4 THEN LET C$ = CHR$ 183
2550 IF C = 5 THEN LET C$ = CHR$ 94
2560 IF C < 6 THEN RETURN
2570 LET C$ = CHR$ (RND* 197 + 33)
2580 IF (CHR$ 144 <= C$ AND C$ <= CHR$ 164)
     OR C$="""" THEN GOTO 2570
2590 RETURN
9000 REM-------------------------------------
9010 PRINT AT 5,1;"HA TENIDO ";C;" ACIERTOS
     SOBRE";
9020 PRINT AT 7,1;"UN TOTAL DE ";N;" TECLAS";
9030 PAUSE 1000
```

Capítulo 3

ESQUEMA DE CONTENIDO

Instrucción LET

Expresiones aritméticas

Representación interna de los datos | Sistema binario

Agrupación de bits

Código ASCII

Casos particulares de las expresiones

Números muy grandes

Números muy pequeños

Funciones intrínsecas

Funciones numéricas

Funciones numéricas y argumento textual

Funciones textuales
- LEFT$
- RIGHT$
- MID$
- Ejemplos de utilización
- Casos particulares
- Funcion VAL$

Funciones textuales y argumentos numéricos

Control de pantalla

3.1 INSTRUCCION LET

En el ZX-Spectrum es obligatorio el uso de la palabra clave LET en la instrucción de asignación. La causa está en que utiliza un teclado con varios modos de funcionamiento. Al principio de la instrucción, el teclado está en modo K y, por tanto, está a la espera de una palabra clave. Puesto que el nombre de una variable empieza con una letra, nos sería imposible escribirla y en su lugar aparecería la palabra clave asociada a dicha tecla.

El LET es necesario

Si intentamos escribir

```
Y=100
```

sin poner la palabra LET delante, veremos que al pulsar la Y aparece la palabra RETURN que es la palabra clave asociada a esta tecla. En cambio, si empezamos escribiendo LET, el teclado cambia a modo de L o C y seguiremos escribiendo el resto de la instrucción sin problemas.

En resumen, en el ZX-Spectrum es obligatorio el uso de la palabra LET.

3.2 EXPRESIONES ARITMETICAS

Para escribir los operadores mantendremos pulsada la tecla SYMBOL SHIFT y a continuación el signo. Los operadores se encuentran sobre las siguientes teclas

+ sobre la K	↑ sobre la H
– sobre la J	(sobre el 8
* sobre la B	) sobre el 9
/ sobre la V	

En él se emplea siempre la flecha vertical (↑) para indicar la potenciación. No existe el acento circunflejo (^). Utilice la tarjeta de localización de teclas para situar los distintos operadores.

La palabra SQR es una función. Para escribirla pasaremos el teclado a modo E. En la tarjeta de localización de las teclas encontrará la posición de todas las funciones. Recordemos que el cambio a modo E se efectúa pulsado CAPS SHIFT y SYMBOL SHIFT simultáneamente. Una vez el teclado esté en modo E pulsaremos la tecla SQR que se encuentra junto con la H.

Realice el cálculo de las expresiones siguientes:

```
PRINT 3+2↑3 - 5 *2        el resultado es 1.
PRINT 3*SQR(16)-SQR(4)     el resultado es 10.
PRINT 12↑2-4              el resultado es 140.
```

3.3 REPRESENTACION INTERNA DE LOS DATOS

3.3.1 Sistema binario

Los ordenadores trabajan todos con el sistema binario. Cuando le entramos un número en decimal, lo traduce a binario antes de trabajar con él. Sin embargo, el ZX-Spectrum permite que le introduzcamos números directamente en binario.

Para que no haya confusiones, el número en binario va precedido de la palabra BIN. Si no fuera así, el número binario 101 se confundiría con el ciento uno en numeración decimal. La palabra BIN se encuentra escrita en verde sobre la tecla B. Para obtenerla hay que poner el teclado en modo E (pulsando CAPS SHIFT mientras mantenemos pulsada SYMBOL SHIFT).

Escribamos el siguiente programa:

```
10 LET A= BIN 1101
20 PRINT A
```

La función BIN para manejar números binarios

Al escribir RUN, el programa imprimirá en pantalla el número 13, que en binario es el 1101. Hemos dado al ordenador un número escrito directamente en binario que es almacenado en A. En la línea 20, el contenido de la variable A es convertido automáticamente a decimal por la instrucción PRINT y presentado en pantalla.

Los números escritos en binario también pueden utilizarse en respuesta a una instrucción INPUT. El siguiente programa nos servirá para practicar la conversión de binario a decimal:

```
10 INPUT "NUMERO BINARIO",B
20 PRINT "EN DECIMAL ES",B
```

Los números que contestaremos a la instrucción de la línea 10 estarán en binario e irán precedidos de la palabra BIN. Por ejemplo, si contestamos BIN 1001, en pantalla aparecerá:

EN DECIMAL ES 9

Como es lógico, detrás de la palabra BIN sólo podemos escribir unos o ceros. Además, si el número es negativo, el signo de negativo debe estar situado antes de BIN. Es decir que para el número −3 en binario no es correcto escribir BIN−11 sino que hay que escribir −BIN 11.

El número total de bits que se pueden escribir es de 16, que corresponde a dos bytes. Si los 16 bits valen uno, el número es 65535.

3.2

3.4 AGRUPACION DE BITS

En el ZX-Spectrum se utilizan 5 bytes para almacenar un número. En estos cinco bytes se almacenan el exponente, el signo y la mantisa; la manera de hacerlo es algo complicada y de momento sólo nos interesa el tamaño que ocupa para poder calcular la memoria que se precisa en las variables de un programa. Las variables textuales ocupan tanta memoria en bytes como longitud de texto.

Distribución de la memoria

El tamaño de la memoria de tipo RAM (la que es utilizable por el usuario) puede ser de 16 Kby (o 16384 by) o de 48 Kby (49152 by), dependiendo del modelo de ZX-Spectrum. Esta memoria se utiliza para almacenar varias cosas:

a) El programa en BASIC escrito por el usuario

b) Las variables definidas en el programa

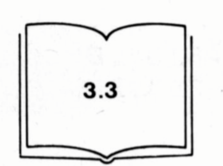
c) Representación visual de la pantalla

d) Gráficos definidos por el usuario

e) Zona de trabajo que necesita el ordenador para realizar cálculos intermedios

El tanto por ciento de memoria que representa cada uno de los apartados depende del programa del usuario, así un programa que tenga muchas instrucciones y pocas variables consume mucho del apartado a y poco del b. Mientras que un programa con muchas variables y pocas instrucciones consume mucho de la parte b y poco de la a.

Sin embargo debe señalarse que siempre se dispone de 16 KB ó 48 KB en total y cada una de las partes citadas se adapta de tal manera que sea posible ahorrar el máximo de memoria.



3.5 CODIGO ASCII

Por ser un código normalizado, el código ASCII no presenta variaciones apreciables respecto a lo que veremos en la lección de programación BASIC en la tabla del código ASCII. Unicamente el código 96, que es un acento grave, se traduce por el símbolo de una libra esterlina (£). Por otra parte, ya sabemos que no utiliza el acento circunflejo, sino la flecha vertical (código 94).

El código ASCII sólo se aplica hasta el 127. A partir de ahí y hasta el 255, cada fabricante utiliza sus propias convenciones. En el caso del ZX-Spectrum, el 127 es el símbolo del «Copyright», que consiste en una © encerrada en un círculo. Del 128 al 143 son símbolos gráficos. Del 144 al 164 son símbolos gráficos definidos por el usuario. Finalmente, del 165 al 255 están codificadas todas las palabras clave del BASIC.

El programa que se da a continuación sirve para visualizar la tabla de códigos que utiliza el ZX-Spectrum.

```
10 LET I = 32
20 PRINT I,"<";CHR$(I);">"
30 LET I = I + 1
40 IF ( I< 256 ) THEN GOTO 20
```

La variable I contiene el número de código, se empieza en 32 porque es el primer carácter imprimible. La línea 10 define la variable I y la inicializa a 32.

La línea 20 escribe el código I y en la columna 17, debido a que se ha colocado un separador coma, se escribe el símbolo <, seguido a continuación del carácter que representa el código, mediante la utilización de la función CHR$ cuyo significado y uso verá más adelante en esta lección, y finalmente el símbolo > para delimitar lo que representa el código. Entre la impresión del símbolo <, el texto del código y el otro símbolo > se colocan puntos y comas para evitar el desplazamiento del cursor.

En la línea 30 se incrementa el valor de la variable I.

Finalmente en la línea 40 se utiliza una instrucción que aún no se ha estudiado, por lo tanto, no se preocupe si no lo entiende, su finalidad consiste en continuar en la línea 20 mientras el código sea menor que 256.

Los símbolos 127 al 255 son especiales para cada ordenador

Al teclear el RUN en pantalla aparecerá la tabla siguiente:

32	< >
33	<!>
.	.
.	.

Los símbolos aparecerán en pantalla hasta llenarla y entonces aparecerá el *scroll*?, tocando una tecla se continuará hasta el código 255, en sucesivas pantallas.

Un problema que puede surgir al manipular textos, es que éstos incluyan el símbolo de las comillas («»), como parte integrante de los mismos. Supongamos que deseamos escribir el siguiente texto:

LA PALABRA «HELLO» SIGNIFICA HOLA

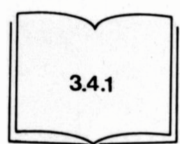
Si escribimos

```
PRINT "LA PALABRA "HELLO" SIGNIFICA HOLA"
```

el BASIC nos dará un error, pues toma como texto a escribir el incluido entre las comillas situadas antes de la palabra LA y las situadas antes de la H de HELLO, con lo que el resto del texto escrito no tiene sentido, dando por tanto un error. Sin embargo, esto puede hacerse.

Para indicar que unas comillas forman parte del texto, se escribirán dos veces seguidas. Así, en el ejemplo anterior, deberíamos escribir:

```
PRINT "LA PALABRA ""HELLO"" SIGNIFICA HOLA"
```



3.6 CASOS PARTICULARES DE LAS EXPRESIONES

El ZX-Spectrum trabaja con una precisión de algo más de 9 cifras (en decimal). De hecho trabaja con la precisión de 32 dígitos binarios, que cuando se transforman a notación decimal representan 9.33 cifras decimales. El significado de una parte fraccionaria de cifra quiere decir que no todos los valores son posibles de los 10 símbolos decimales.

Las cifras escondidas

En el momento de escribir los resultados sólo muestra 8 cifras exactas, pero internamente mantiene al menos 9 cifras. El siguiente ejemplo es ilustrativo:

```
10 LET A = 1234554321
20 PRINT A,A-123E+7
```

Al escribir RUN, el programa da dos resultados:

1.2345543E+9 y 4554321

El primer resultado es el contenido de la variable A. Observamos que, aparentemente, se han perdido las dos últimas cifras. Sin embargo, el segundo resultado demuestra que internamente existían, puesto que al realizar la resta

1234554321 − 1230000000

hemos hecho desaparecer las tres primeras cifras, y ahora el ZX-Spectrum nos muestra las dos cifras escondidas del final.

3.7 NUMEROS MUY GRANDES

En el ZX-Spectrum el mensaje de «Overflow» del BASIC estándard está cambiado por:

6 Number too big

que significa «número demasiado grande». Si efectuamos el programa indicado, veremos que este mensaje se produce cuando el valor de A supera el número 10 elevado a la potencia 38.

El hecho de que los ordenadores trabajen en binario con una precisión limitada, conduce a veces a resultados paradójicos, cuando se mezclan números muy grandes con números pequeños. Probemos el siguiente caso:

```
PRINT 1E+10 + 1 - 1E+10   da 0 como resultado
PRINT 1E+10 - 1E+10 + 1   da 1 como resultado
```

El efecto del número limitado de cifras del ordenador

¡Dos cálculos aparentemente idénticos dan resultados distintos! ¿Cómo es posible este hecho?» Desde el punto de vista matemático da igual sumar y luego restar que el proceso inverso. Cuando se trabaja con un número de cifras limitado esto puede no ser cierto, y éste es precisamente el caso que se presenta.

Como ya sabemos, las expresiones se evalúan de izquierda a derecha. En el primer caso, sumamos la unidad 1E+10 (un uno seguido de diez ceros). Puesto que se supera la precisión de la máquina, sólo se almacena 1E+10, pero no el 1 que sumamos. A continuación se le resta 1E+10, con lo que el resultado global es 0. Por el contrario, en el segundo caso, se efectúa primero la resta de 1E+10 menos 1E+10, cuyo resultado es 0, y a continuación se le suma una unidad. Por lo tanto el resultado final obtenido es 1.

Para verlo de un modo más gráfico escribamos la cantidad en notación de todas las cifras; es decir

10000000000

al sumarle 1 queda

10000000001

si la máquina retiene 10 cifras lo que se ha memorizado es 100000000? que son 10 cifras y el interrogante indica que detrás del interrogante supondrá que viene un cero aun cuando había un 1. Al restarle otra vez 10000000000, dará cero pues no ha podido memorizar el último 1.

Por el contrario, al restar primero 10000000000 menos 10000000000 dará cero aunque tome sólo 9 dígitos y después de sumarle 1 sí lo tiene en cuenta, puesto que 0 + 1 sí cabe en el tamaño de cifras con que opera.

Este comportamiento no debe hacernos perder la confianza en la capacidad de cálculo de los ordenadores. Es simplemente una consecuencia de la limitación del número de cifras. Las calculadoras de bolsillo tienen exactamente el mismo problema.

Durante la conversión de decimal a binario se efectúan redondeos. Estos redondeos pueden originar comportamientos aún más sorprendentes. Probemos el siguiente ejemplo:

```
PRINT 5E+9 + 1 - 5E+9
```

¡El resultado es 2!

Ello se debe a que estamos trabajando en el límite de la precisión (el valor 5E+9 se ha escogido a propósito). Al sumar una unidad, se redondea por encima, quedando el número 5000000002. Al restarle 5E+9 se obtiene 2 de resultado.

Para aclarar un poco este hecho, vamos a considerar las operaciones en binario; hay que advertir que el ordenador no lo hace exactamente así pero que la idea es correcta.

En primer lugar, calculemos cómo es el número 5E+9 en binario, usted puede calcularlo, es un poco largo pero es un buen ejercicio.

1|00101010|00000110|11110010|00000000

El redondeo en binario

Las barras se colocan para separar claramente los grupos de 8 bits y hacer el número más fácilmente legible. Como puede ver ocupa exactamente 33 dígitos binarios; sin embargo, los últimos dígitos son cero. Al memorizarse este número, se ignora la última cifra binaria y se le supone cero, de tal manera que sólo se almacena,

|10010101|00000011|01111001|00000000|

Las cifras que se suponen detrás de la última barra son ceros, y que en nuestro caso corresponden a la realidad. (Sólo supone un cero.)

Cuando el ordenador realiza la operación de sumar 1 se da cuenta que debe sumar en la posición inmediatamente a la derecha de la última barra, como presupone que hay un cero en el número grande, sabe que el resultado en esta cifra que no existe sería un 1. Entonces entra en marcha el mecanismo de redondeo, y como esta cifra contiene un 1 añade un 1 a la siguiente de la izquierda pues el resultado correcto supone que está más cerca de la realidad si coloca este 1. Por lo tanto, el resultado queda como

|10010101|00000011|01111001|00000001|

Entonces en el momento de convertir el número a decimal sabe que debe añadir un cero a la derecha de la última barra, por lo tanto (hágalo usted mismo), el resultado será de

5000000002

pues ha convertido el número en binario

1|00101010|00000110|11110010|00000010

Si a este número le restamos 5000000000, evidentemente nos queda un dos.

El mecanismo de suponer lo que hay después de la columna de la derecha y de tomar la decisión de arrastrar un 1 hacia la izquierda después de considerar lo que debería de haber en la columna de la derecha es el mecanismo de redondeo binario.

De hecho toda la teoría de los sistemas de numeración y la precisión es una disciplina matemática muy importante en las cuestiones numéricas; sin embargo, en la mayoría de aplicaciones prácticas el ordenador se comporta según lo que esperamos. Le mostramos estos hechos para que recuerde que en alguna ocasión cuando la fuente de los problemas en un cálculo no está clara debe comprobarse que no suceden cosas de este estilo. Por lo demás puede prescindir de estas explicaciones si no tiene especial curiosidad por ellas.

3.8. NÚMEROS MUY PEQUEÑOS

En el ZX-Spectrum el número más pequeño que se puede manejar es el $4*10^{-38}$.

En el ZX-Spectrum no existe limitación en el número de símbolos que puede contener un texto. Obviamente existe el límite de la capacidad total

de la memoria de la máquina. No obstante, se recomienda, siempre que sea posible, no sobrepasar el tope de 255 símbolos a fin de que nuestros programas no sean incompatibles con los demás ordenadores.



3.9 FUNCIONES INTRINSECAS

En el teclado del ZX-Spectrum las funciones se encuentran escritas fuera de las teclas. Para utilizarlas hay que pasar al modo de operación extendido (E). Para ello, mantendremos apretada la tecla SYMBOL SHIFT y pulsaremos CAPS SHIFT. En el cursor aparece la letra E parpadeante, indicativa del modo extendido. Entonces, al pulsar la tecla elegida aparecerá escrita la función y el teclado volverá a modo C o L.

Se pueden suprimir los paréntesis

Una diferencia importante del BASIC del ZX-Spectrum es que no es imprescindible que el argumento vaya entre paréntesis. Las funciones se comportan como operadores unarios de máxima prioridad. Por ejemplo

```
PRINT SQR 4 + 1
```

En primer lugar se evalúa la raíz cuadrada de 4, ya que la función SQR tiene menor nivel que el signo más (+). Al resultado —que es 2— se le suma la unidad, con lo que se obtiene 3 como resultado final.

Aunque en él no son necesarios los paréntesis, no recomendamos omitirlos por dos razones. La primera es que los paréntesis hacen más legibles las expresiones y, además, su empleo impedirá que cometamos errores al considerar el orden de prioridad en una expresión compleja.

De esta forma, la expresión anterior quedaría más clara escrita de la forma

```
PRINT SQR(4) + 1
```

Por otra parte, a veces es estrictamente necesario escribirlos. Por ejemplo, en la expresión

```
PRINT SQR(16+9)
```

no se pueden omitir, ya que lo que se desea es la raíz cuadrada de la suma de 16 más 9, es decir, la raíz cuadrada de 25 que es 5; por tanto SQR se debe aplicar sobre la suma de ambos. En este caso, si no se pusieran los paréntesis, el ordenador haría primero la raíz cuadrada de 16 que es 4 y lo sumaría a 9, con el resultado de 13.

La segunda razón es que la gran mayoría de variantes del BASIC requieren el uso de paréntesis con las funciones. Si estamos acostumbrados a usarlos, nos será más fácil entender un listado de un programa escrito

para otro ordenador. De esta forma, no tendremos dificultad en trasladar programas de otras máquinas, y adaptarlos para que funcionen con el ZX-Spectrum.

3.10 FUNCIONES NUMERICAS

No hay apenas variaciones entre las funciones numéricas estándar del BASIC y las que utiliza el ZX-Spectrum. La única diferencia está en la función LOG, que en él se denomina LN (de Logaritmo Natural). Esta función se encuentra sobre la tecla correspondiente a la letra Z. Aparte del cambio de nomenclatura, el funcionamiento de LN es idéntico al de LOG.

```
PRINT LN(2.718282) escribe 1.0000001
```

Aparte de las funciones estándar, el ZX-Spectrum dispone de algunas más. Las dos primeras son de índole matemática, por lo que únicamente las citamos a fin de dar una visión completa y son:

ACS(X) Función trigonométrica que calcula el arco coseno de X. El resultado está en radianes. El valor del argumento debe estar comprendido entre −1 y +1. Se encuentra sobre la tecla de la letra W.

Por ejemplo,

```
PRINT ACS(0.5)
```

escribe 1.0471976.

ASN(X) Función trigonométrica que calcula el arco seno de X. El resultado está en radianes. Se produce un error si el argumento X no está en el margen de −1 a +1. Se encuentra escrita sobre la tecla Q.

Por ejemplo,

```
PRINT ASN(0.5)
```

escribe 0.52359878

La tercera, más que una función, es el almacenamiento de un número que seguramente conoce y que tiene importancia en el cálculo de perímetros, superficies y volúmenes de figuras con partes circulares, es el número PI (π).

PI Esta función no tiene ningún argumento. El resultado es el número PI (π), igual a 3.1415926536... Se encuentra sobre la tecla M. Por ejemplo, para calcular la longitud de una circunferencia cuyo radio es 8, escribiremos

```
NEW
10 LET R = 8
PRINT 2*PI*R
```

Al teclear RUN el resultado que aparecerá es 50.265482.

3.6

3.11 FUNCIONES NUMERICAS Y ARGUMENTO TEXTUAL

En el ZX-Spectrum, la función estándar ASC que da como resultado el valor del código ASCII del argumento, se denomina CODE, y se encuentra sobre la tecla correspondiente a la letra I. El funcionamiento es idéntico al de la función ASC.

```
PRINT CODE("A") escribe 65
```

La función VAL tiene una interesante ampliación respecto al BASIC estándar. Sabemos que el argumento es un texto con significado numérico. En el caso del ZX-Spectrum, se admite, además, que el argumento sea una expresión. Por ejemplo, si escribimos

```
LET A = VAL("1+3")
PRINT A
```

el resultado será 4. En el BASIC estándar el resultado hubiera sido de cero o un error, ya que «1+3» no es un número, sino una expresión. Incluso podemos escribir:

```
PRINT VAL("3"+"-2")
```

La función VAL es muy potente

cuyo resultado es 1. En efecto, la primera operación es unir «3» con «−2», ya que, como sabemos, el signo más (+) es el símbolo de concatenación. El resultado de esta primera operación es pues «3−2». La función VAL actúa sobre este resultado, obteniendo 1 como resultado final.

A pesar de que esta función es mucho más potente en el ZX-Spectrum, hay que procurar no hacer un abuso de las posibilidades no estándar. En caso contrario, corremos el riesgo de realizar programas totalmente incompatibles con el resto de ordenadores.

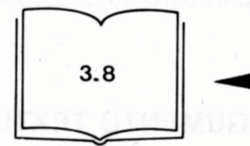
Si el argumento no tiene significado numérico, o de expresión numérica, se produce un error. Por ejemplo,

```
PRINT VAL("1A")
```

dará el mensaje de error:

C : Non sense in BASIC

que significa que no tiene sentido en BASIC.



3.12 Funciones textuales

En el ZX-Spectrum no existen las funciones estándars de fragmentación de textos LEFT$, MID$ y RIGHT$. En su lugar, emplea el llamado operador de fragmentación o de subcadena. Recordemos que un dato textual se llama también a veces «de cadena», por estar formado por una cadena de símbolos. El operador de subcadena es un medio más potente que las funciones estándars para extraer trozos de textos. Muchas variantes de BASIC de ordenadores grandes incorporan este operador como alternativa de las funciones antes reseñadas.

El operador de fragmentación tiene la forma general siguiente:

(inicio TO final)

El operador de fragmentación

inicio y final son dos expresiones numéricas. La primera de ellas indica dónde comienza a fragmentarse el texto, y la segunda, dónde termina. La palabra TO se encuentra señalada en la tecla correspondiente a la letra F. Este operador se sitúa a la derecha de un dato o expresión textual. Por ejemplo, escribamos:

```
10 LET A$ = "CASA"
20 PRINT A$(2 TO 4)
```

El resultado después de teclear RUN es ASA, ya que se extrae desde la segunda letra hasta la cuarta. La palabra TO significa «hasta» en inglés. Por tanto, esta operación se leería como «A$ desde dos hasta cuatro». Hagamos algunas pruebas más. Escribamos primero (conservando la sentencia 10):

```
20 LET B$ = "ABCDEFG"
```

A continuación probemos las siguientes instrucciones:

```
30 PRINT B$(1 TO 3) escribe ABC
40 PRINT B$(3 TO 6) escribe CDEF
50 PRINT B$(6 TO 7) escribe FG
60 PRINT A$ + B$(1 TO 4) escribe CASAABCD
```

Al teclear, el RUN aparecerá en pantalla

ABC

CDEF

FG

CASAABCD

Este último resultado puede parecernos sorprendente a primera vista. En primer lugar debe tenerse en cuenta que no se trata de una función sino de un operador. Sucede que este operador es de menor nivel que el operador de concatenación (+). Por tanto, se efectúa primero la fragmentación (1 TO 4), cuyo resultado es ABCD. Sobre él actúa el operador de concatenación, con lo que se obtiene, efectivamente, CASAABCD.

La alternativa es (A$ + B$) (1 TO 4) que primero une A$ con B$ para dar «CASAABCDEFG» y de este texto toma las cuatro primeras letras; por lo tanto, el resultado es «CASA».

Existe una analogía entre este operador de subcadena o de fragmentación, y las funciones LEFT$, MID$ y RIGHT$. Vamos a verlo en las siguientes secciones.

3.12.1 LEFT$

Para producir el mismo efecto que la función LEFT$, aplicaremos la regla siguiente:

```
LEFT$(X$,N) es equivalente a X$ (1 TO N)
```

Así, por ejemplo

```
PRINT LEFT$("CASA".2)
```

se escribirá en el ZX-Spectrum

```
PRINT "CASA"(1 TO 2)
```

y en la pantalla aparecerá CA.

El operador de fragmentación es muy flexible y permite omitir alguna de sus partes. De esta forma, si se omite el inicio, él supone que se empieza desde 1. Por tanto, se puede simplificar la regla anterior escribiendo

```
LEFT$(X$,N) es equivalente a X$ ( TO N)
```

y en el ejemplo, podremos escribir, obteniendo el mismo resultado:

```
PRINT "CASA"(TO 2)
```

3.12.2 RIGHT$

Para extraer la parte derecha de un texto, escribiremos el operador de la forma siguiente:

(N TO)

es decir, omitiendo el final. En este caso, se obtiene la subcadena desde el carácter N hasta el último. Por ejemplo:

```
PRINT "CASA"(2 TO)
```

da como resultado ASA. Nótese que, a diferencia de la función LEFT$, no son equivalentes los valores de la N del operador y de la N de la función RIGHT$. Por ejemplo, en el BASIC estándar

```
PRINT RIGHT$("CASA",2) escribe SA
```

Esto es debido a que en este caso, la N significa los N últimos. Para convertir «los N últimos» en «desde N hasta el final» hay que hacer la operación

```
LEN(X$)-N+1
```

Como utilizar la función RIGHT

Entonces, para producir exactamente el mismo efecto que la función RIGHT$, aplicaremos la regla siguiente:

```
RIGHT$(X$,N) es equivalente a X$
((LEN(X$)-N+1) TO)
```

Por ejemplo,

```
10 LET X$ = "Programacion"
20 PRINT X$((LEN(X$)-3+1) TO)
```

en pantalla aparecerá ion.

Ciertamente esta regla es algo complicada. No recomendamos su uso excepto en el caso de que deba adaptarse al BASIC de un programa escrito en otro ordenador e, incluso en este caso, casi siempre es más rentable realizar pequeños retoques en el programa, a fin de aprovechar todas las posibilidades del operador.

3.12.3 MID$

Para extraer la parte central de un texto, se escribe el operador de forma completa, es decir (N TO M). Por ejemplo:

```
PRINT "CASA"(2 TO 3) escribe AS
PRINT "CASA (3 TO 3) escribe S
```

En este último ejemplo, vemos además la forma de extraer una sola letra, escribiendo el mismo número a ambos lados de la palabra TO. La regla de equivalencia de la palabra TO es:

```
MID (X$,N,M) es equivalente a X$
(N TO (N+M-1))
```

Vamos a ver algunos ejemplos de utilización del operador.

3.12.4 Ejemplos de utilización

Un programa de descomposición de la fecha, se escribirá:

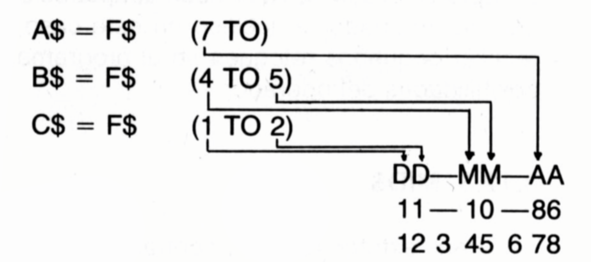
```
NEW
10 REM Descompactacion de la fecha
20 INPUT "FECHA (DD-MM-AA): ",F$
30 LET D$ = F$(1 TO 2)
40 LET M$ = F$(4 TO 5)
50 LET A$ = F$(7 TO)
60 PRINT D$;" DEL ";M$;" DE 19";A$
```

En las líneas 30, 40 y 50 se sustituyen las funciones LEFT$, MID$ y RIGHT$ por el operador equivalente.

Ejecutemos el programa, dando, por ejemplo, la fecha 11-10-86, entonces la variable F$ contiene:

```
FS = 1 1 – 1 0 – 8 6
     | | | | | | | |
     D D – M M – A A
```

al descompactar, hemos realizado estas operaciones:



Como es fácil de apreciar es mucho más cómodo para programar utilizar el operador que no las funciones estándar.

El programa para ver qué letra ocupa una posición determinada, la 1, la 2, etc., se escribirá:

```
NEW
10 INPUT "FRASE: ",A$
20 INPUT "LETRA EN LA POSICION: ",P
30 PRINT A$(P TO P)
40 GO TO 20
```

Al ejecutarlo entremos la FRASE, *Hola que tal.* Al entrar la P, podremos averiguar cuál es la letra 6, que debe dar q. Cuál es la letra 9? Es un espacio en blanco. Para terminar el programa utilice el STOP cuando le pida la LETRA EN LA POSICION.

3.12.5 Casos particulares

Ya hemos mencionado que en el operador de fragmentación se pueden omitir algunas partes. Según sea la parte que se omita, el significado es uno u otro. Supongamos un texto de longitud L:

1/ (TO N) equivale a (1 TO N)

2/ (N TO) equivale a (N TO L)

3/ (TO) equivale a (1 TO L)

4/ (N) equivale a (N TO N)

5/ () equivale a (1 TO L)

Veamos el ejemplo siguiente,

```
LET A$ = "Programacion"
LET L = LEN (A$)
```

que se ejecutan en modo inmediato, la función LEN la veremos en el apartado siguiente y da la longitud de A$. Ahora ejecutemos también en modo inmediato las instrucciones siguientes:

```
PRINT A$(TO 3)    escribira Pro           equivalente a A$(1 TO 3)
PRINT A$(3 TO)    escribira ogramacion    equivalente a A$(3 TO L)
PRINT A$(TO)      escribira Programacion  equivalente a A$(1 TO L)
PRINT A$(3)       escribira o             equivalente a A$(3 TO 3)
PRINT A$()        escribira Programacion  equivalente a A$(1 TO L)
```

Ciertamente las equivalencias 3 y 5 no tienen más valor que el anecdótico. En cambio, la 4 tiene interés, puesto que es una simplificación para obtener una sola letra

```
PRINT "CASA"(3) escribe S
```

en lugar de escribir «CASA» (3 TO 3).

El operador de fragmentación da un error si alguno de los números está fuera de margen (mayor que la longitud o menor que cero). Sin embargo, si el primer valor es mayor que la longitud del texto, el resultado es la cadena vacía.

```
PRINT "CASA"(3 TO 8) de el error 3 subscript wrong
PRINT "CASA"(8 TO 10) da la cadena vacia 8 es mayor que longitud
PRINT "CASA"(1 TO 0) da la cadena vacia 0 es menor que  1
```

El error subscript wrong significa que el 8 está más allá de la longitud del texto CASA.

El operador de fragmentación no sólo se puede utilizar para extraer trozos de textos, sino también para realizar asignaciones. Por ejemplo, escribamos:

```
10 LET A$ = "CASA"
20 LET A$(1 TO 3) = "ROM"
30 PRINT A$
```

El resultado es ROMA, puesto que hemos sustituido las tres primeras letras de CASA por las letras ROM. Si cambiamos la línea 20 por:

```
20 LET A$(1 TO 3) = "ROMPER"
```

el resultado será el mismo ROMA, puesto que sólo se toman las tres primeras letras de ROMPER.

Si se escribe

```
20 LET A$(1 TO 3) = "RO"
```

el resultado es RO A. Si al realizar una asignación de subcadena el texto es demasiado corto, el BASIC lo rellena con espacios en blanco. Por eso aparece un espacio en blanco entre RO y A.

Como práctica del operador de cadena, escribamos el siguiente programa para escribir el día de la semana:

```
10 REM Dia de la semana
20 LET A$= "LunMarMieJueVieSabDom"
30 INPUT "Dia (1 a 7): ",D
40 LET B = (D-1)*3+1
50 PRINT A$(B TO B+2)
```

En la línea 20, se construye un texto formado por las tres primeras letras del nombre de los días. En la línea 30 se pide que el operador entre un número comprendido entre 1 y 7. En la línea 40 se calcula la posición donde empieza el nombre del día. Así, si D vale 4 el resultado de B será (4−1)*3+1 = 10 que corresponde a la letra J. El número 3 se usa porque tomamos las tres primeras letras del nombre del día. Si se hubieran escogido 5 letras, la fórmula sería:

```
40 LET B = (D-1)*5+1
```

En la línea 50, se escribe desde la posición calculada hasta 2 letras más (en total 3 letras). Si D valía 4, el resultado es Jue. Es decir, del texto A$(«LunMarMieJueVieSabDom») se toma desde el valor B (que era 10) hasta dos letras más. Por lo tanto, J hasta e (Jue).

3.12.6 Función VAL$

En el ZX-Spectrum existe la función VAL$, no demasiado útil. El argumento es de tipo textual, tal como indica el símbolo dólar ($) al final del nombre. Está situada en la letra J.

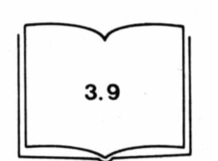
La función consiste en eliminar las comillas que puedan haber en el texto. Tenga en cuenta que las comillas en el interior de un texto deben ser dobles.

Sin embargo, no recomendamos la utilización de esta función por dos razones:

a) Porque es una función muy específica del Spectrum y, por tanto, un programa que la utilice no se adaptará fácilmente a otro BASIC.

b) Porque, cuando el argumento no contiene comillas adicionales, el resultado es un error.

En definitiva, que es una función cuya no utilización no le va a impedir en ningún momento escribir los programas que desea y cuya utilización sí le puede acarrear problemas.



3.13 FUNCIONES TEXTUALES Y ARGUMENTOS NUMERICOS

No hay variación respecto a las dos funciones estándar del BASIC CHR$ y STR$. Por el contrario, la función RPT$ no existe en el Spectrum. Para reproducir el efecto de la función RPT$, utilizaremos un texto entre comillas formado por el número de símbolos indicado en RPT$. Si el número de símbolos se calcula durante la ejecución del programa, no se puede sustituir directamente. Para ello utilizamos un bucle de repetición controlada que veremos en lecciones posteriores.



3.14 CONTROL DE PANTALLA

La instrucción CLS se encuentra sobre la tecla correspondiente a la letra V. Su funcionamiento es el estándar.

La función de control de posición AT situada en la letra I, tiene una particularidad. En el ZX-Spectrum tiene la forma

AT F,C

No hay que poner paréntesis en el AT

es decir, primero se escribe el valor correspondiente al número de línea o fila. Por otra parte, esta función no debe llevar paréntesis. El ángulo superior izquierdo de la pantalla es la posición 0,0. La pantalla del Spectrum tiene 32 columnas de ancho (de 0 a 31) y 22 líneas de alto (de 0 a 21). En la zona de trabajo (líneas 22 y 23) no se puede escribir, puesto que están reservadas para la entrada de datos u órdenes y para los mensajes del ordenador. La retícula de la pantalla para el ZX-Spectrum está dibujada en la figura 1.

Para colocar un asterisco (*) en la parte central de la pantalla escribiremos:

```
PRINT AT 11,16;"*"
```

Los valores de las coordenadas deben estar comprendidos entre los límites indicados. De lo contrario, se produce un error, con el mensaje

Out of screen o bien *Integer out of range*

que indica fuera de pantalla

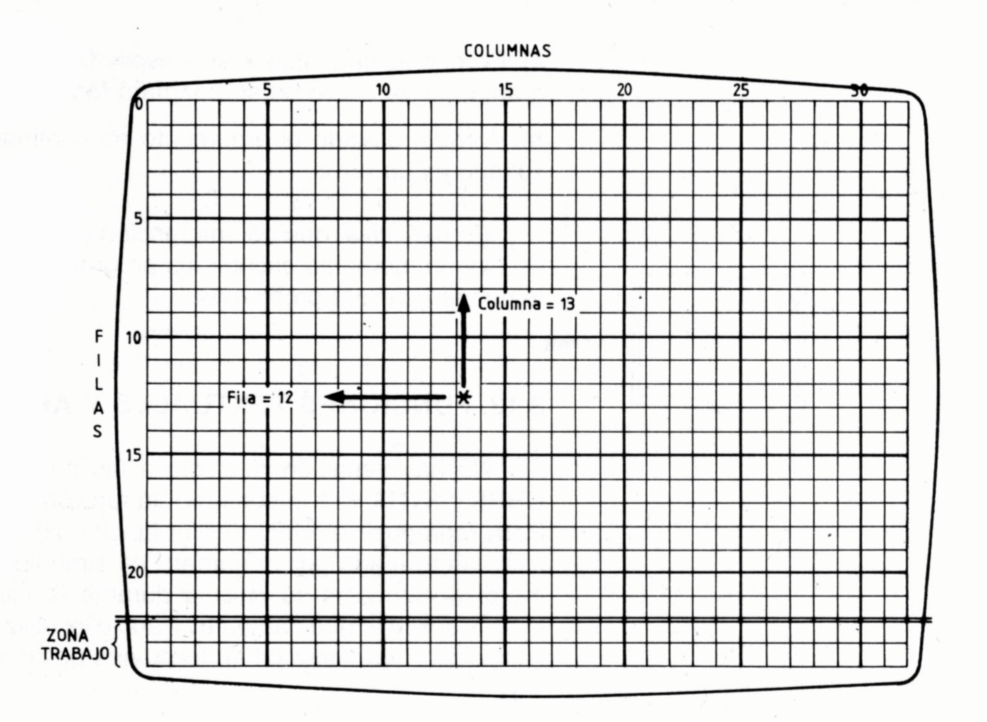


Figura 1 Retícula de posicionado en pantalla

Para probar la función AT escribamos el siguiente programa. La función del mismo es mover un asterisco (*) horizontalmente por la pantalla. Para ello, utilizaremos la función AT dentro de un bucle. Escribiremos el asterisco cada vez más hacia la derecha. Simultáneamente, borraremos los asteriscos que quedan a la izquierda, a base de escribir espacios en blanco

```
10 REM Movimiento horizontal
20 LET Y = 11 : LET X = 1
30 PRINT AT Y,X;"*"
40 LET X = X+1
50 PRINT AT Y,X;"*";AT Y,X-1;" "
60 GOTO 40
```

En la línea 20 establecemos los valores iniciales de fila y columna. En la 30 situamos el asterisco en la posición inicial. A continuación, en la línea 40, se incrementa el número de la columna (avance hacia la derecha). En la línea 50 se escribe el asterisco (*) en la nueva posición calculada y simultáneamente se borra el asterisco anterior (posición X-1). Finalmente, la línea 60 transfiere de nuevo el control a la línea 40. El programa se detiene cuando el asterisco se sale de la pantalla, y se produce un error.

Probablemente, notaremos que el asterisco se desplaza demasiado rápidamente para observarlo. Para hacer su marcha más lenta, incluiremos dentro del bucle un cálculo inútil, que haga perder tiempo al ordenador a fin de retardar el avance. Escribiremos la línea

```
45 LET A = SQR(99)
```

Al ejecutar el programa, observaremos que el asterisco se desplaza mucho más lentamente.

Si vuelve a mirar el programa que entró como práctica para estudiar el teclado en la unidad didáctica 2, en la lección «Prácticas con su microordenador», observará la utilización del AT en muchas de sus líneas.



Capítulo 4

ESQUEMA DE CONTENIDO

Operadores relacionales

Función MOD

Prácticas	Práctica 1 Práctica 2 Práctica 3

4.1 OPERADORES RELACIONALES

En el SPECTRUM, los operadores relacionales se escriben pulsando una sola tecla aunque consten de dos símbolos como <= o >=.
Estos símbolos se encuentran escritos en las siguientes teclas

=   sobre la L

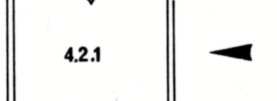
<   sobre la R

>   sobre la T

<= sobre la Q

>= sobre la E

<> sobre la W



4.2 FUNCION MOD

La función MOD no está incluida en el catálogo de funciones del SPECTRUM. Es una función aritmética que calcula el resto de una división. Por ejemplo, la función MOD(14,4) realiza la operación de dividir 14 por 4, pero el resultado que nos da no es la división, sino el resto, una vez quitada la parte entera de la misma. En este caso, el resto de la división es 2. Esta función se representa de modo general como:

MOD(X, Y)

en donde X es el dividendo e Y el divisor. Ya hemos dicho que el resultado que calcula es el resto.

En el SPECTRUM no existe esta función MOD. ¿Cómo hacer, pues, el cálculo de la función MOD con el SPECTRUM?

Vamos a verlo. Tenemos para ello otra función, la función INT que lo que calcula no es el resto, sino el cociente (parte entera) de la división. Es decir que INT(14/4) nos da como resultado 3, o lo que es lo mismo, la parte entera de esta división.

Tal como se vio en la lección 3, en las funciones numéricas, la función INT se representa:

INT(X/Y)

en donde X es el dividendo e Y el divisor. Insistimos en que el resultado es la parte entera de la división. (No escriba nada hasta que llegue el programa en el que la aplicaremos.)

Sabiendo esto, podemos construir una fórmula que nos calcule lo mismo que calculaba la función MOD. Esta fórmula es la siguiente:

X–INT(X/Y)*Y

Ahora vamos a aplicar los mismos valores a X (dividendo) y a Y (divisor) y veremos que nos da el mismo resultado:

X = 14

Y = 4

INT(X/Y) = 3

Por tanto:

14 – 3*4 = 2

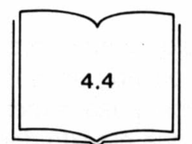
Según lo anterior, la línea 20 del programa que escribirá en seguida será:

```
20 LET DS=DS+1 : LET SS=DS-INT(DS/7)*7
```

El programa completo sería:

```
NEW
10 INPUT DS : Se entra el codigo del dia
              de la semana.
20 LET DS= DS+1 : LET SS=DS-INT(DS/7)*7
30 PRINT SS
40 GOTO 10
```

Ahora deje escrito el programa y antes de ejecutarlo continúe estudiando la lección de programación.



4.3 PRACTICAS

Con el fin de hacer una recapitulación de todos los comandos e instrucciones estudiadas hasta ahora, le proponemos realizar estas tres prácticas. Éstas consisten en hacer tres programas que contienen, desde un punto de vista práctico, los elementos que hemos utilizado en el aprendizaje de la programación.

Antes de entrar en el contenido de estos programas, explicaremos el método que seguiremos para elaborarlos. Lo hacemos así para que vea la manera cómo procede un programador profesional. Los programas concretos que realizará son relativamente cortos y de una utilidad limitada. Pero contienen los distintos pasos a seguir en su construcción, y que son los que utiliza el programador en la realización de programas mucho más largos y de utilidad práctica.

Lo primero que debe hacerse, antes de iniciar un programa, es definir lo más claramente posible el problema a resolver. A esto se le denomina

definir las especificaciones del programa. En este paso se hace un listado de todos los requisitos que debe cumplir un programa para resolver el problema que se plantea.

En un segundo paso hay que *analizar todos los datos* que son necesarios para la realización del programa. Estos son de tres tipos:

— Los que son necesarios como entrada de datos.

— Los que son necesarios para realizar los cálculos.

— Finalmente, los que son necesarios para emitir el resultado del problema.

Tanto el paso primero como el segundo deben realizarse antes de comenzar la verdadera tarea de programación. Ya le advertimos, que cuanto más tiempo se dedique a comprender y diseñar los objetivos y los datos que intervienen en la solución del problema, tanto más rápida y libre de errores será la programación.

Tenga en cuenta que el requisito más importante de un programa es que haga la tarea correctamente y de acuerdo con los requisitos que se han fijado. Un programa erróneo es una creación que induce a más errores y que puede darnos más problemas que si la solución la hubiéramos hecho a mano directamente.

Insistimos en que en el mundo de la programación se tiene muy claro que un programa es tanto mejor cuanto más tiempo se dedique a elaborar estos dos pasos indicados, y cuanto más tardemos en escribir en lenguaje de programación.

Refrene siempre la impaciencia de querer escribir directamente un programa. Si queremos resolver un problema escribiéndolo directamente, lo más probable es que tengamos que hacer muchas correcciones, que oscurecen el significado. Las ideas afloran delante del teclado de modo desordenado y el texto resultante será tan enredado que ni nosotros mismos lo entenderemos al cabo de poco tiempo.

Si queremos escribir un texto para un artículo, primero ordenamos lo que queremos decir en forma de índice a fin de escribir después con una estructura clara. En el caso de la programación pasa exactamente lo mismo; se deben dar los pasos estudiados para que el texto del programa se adapte perfectamente a la estructura del problema a resolver.

Una vez que hemos resaltado la importancia de los dos pasos primeros, vamos a ver el tercer paso, que consiste en *escribir el texto del programa* en el teclado. Además de numerar las líneas de forma espaciada (por ejemplo, de 10 en 10), es recomendable dejar espacios de numeración entre los bloques del programa, con el fin de facilitar los últimos retoques.

Finalmente, el cuarto paso consiste en *probar el programa,* con unos datos que sepamos de antemano el resultado para ver si el programa responde según esperamos. De todas maneras, debe tener en cuenta que, aunque un programa supere las pruebas correctamente, no se demuestra con ello que el programa funcione correctamente. Las pruebas sólo sirven para evidenciar errores, pero no para demostrar que un programa es correcto. (No se asuste por lo que acabamos de decir y no por ello debe perder la confianza en los ordenadores. Sólo le hemos querido indicar, diciendo las cosas como son, la necesidad de probar los programas.)

En resumen, la elaboración de un programa consiste en realizar estos cuatro pasos:

— Definir el problema y los requisitos.

— Diseñar con qué datos trabajaremos.

— Teclear el texto del programa.

— Hacer pruebas para poner en evidencia los errores.

Pasemos ahora a resolver las tres prácticas que le proponemos.

4.3.1 Práctica primera

Paso 1: Definición del problema

Vamos a construir un programa para que realice la factura que presentamos en la unidad didáctica 1. Queremos resaltar que todavía no es un programa para hacer facturas en general, sino única y exclusivamente aquella factura. El programa resultante sabemos que no es demasiado útil, pues los programas son precisamente para casos más generales. (Ya llegaremos también a ello.) Nuestra finalidad en este momento es didáctica y hay que ir poco a poco.

Conviene que consulte Vd. el capítulo 1 de este libro y busque la figura 1 para tenerla presente durante la realización de esta práctica.

Los requisitos que impondremos al programa son:

— El soporte de salida será la pantalla y no papel. Por tanto, el resultado lo podremos comprobar en la pantalla.

— No es necesario que aparezca la parte de la factura que está con la letra de imprenta, sino sólo lo que está a mano. Evidentemente, si la salida fuera por papel, el papel ya llevaría esto impreso.

— Sin embargo, hay que tener en cuenta el encolumnado que daremos en pantalla, pues difiere de la anchura del papel. Ya sabe que en el caso del SPECTRUM disponemos de 32 columnas. Dejaremos 15 para el artículo; después una columna en blanco (la 16); de la 17 a la 20 colocaremos la cantidad; la 21 la dejamos en blanco; de la 22 a la 25 colocaremos el precio unitario; la 26 la dejamos en blanco y desde la 27 a la 31 colocamos el importe. Resumiendo:

Columnas

1–15	Artículo
16	En blanco
17–20	Cantidad
21	En blanco
22–25	Precio unitario
26	En blanco
27–31	Importe

Por otra parte, la fecha la empezaremos a partir de la columna 9 y la dirección de la factura, que constará de tres líneas, ocupará las columnas 16 a 31.

Paso 2: Los datos

Los datos que debemos dar al problema son:

a) La fecha (F$)

b) Primera línea de la Dirección (D$)

c) Segunda línea de la Dirección (E$)

d) Tercera línea de la Dirección (G$)

e) Para cada uno de los artículos hay que entrar:

e.1) Nombre (o$, p$, q$)

e.2) Cantidad (c1, c2, c3)

e.3) Precio unitario (p1, p2, p3)

e.4) Importe (i1, i2, i3)

f) El tanto por ciento de impuesto que debemos cargar (tpc)

Habrá observado que entre paréntesis ya estamos concretando qué nombre daremos a las variables que utilizaremos.

El proceso lo dividiremos en dos partes: Entrada de datos y emisión de la factura.

En la entrada mantendremos un reflejo en pantalla de los datos que se han ido entrando para recordar los anteriores, cuando vayamos a entrar otro.

Para emitir el resultado, será necesario ir calculando el total acumulado (t).

En este momento es necesario que revise la lista de variables que ha utilizado para comprobar que no hay repeticiones.

Ahora que debe empezar enseguida a escribir el programa, le pedimos todavía un poco de paciencia y que tome un papel y lápiz antes de que comencemos directamente con el teclado. Usted debería confeccionarse un esquema similar al que le proponemos, a la vista de la figura 1 del primer capítulo de este volumen. (Este esquema no contiene propiamente instrucciones de programación, pero le da una idea de cómo será el programa y de qué estructura tendrá).

1) *Fase de entrada:* Deberá recoger todos los datos y los irá imprimiendo en pantalla para recordarlos. Las instrucciones tendrán la forma de:

```
INPUT «Fecha: »; f$: PRINT «Fecha: »; f$
```

Habrá que repetir instrucciones de este tipo para todos los datos a entrar:

— Línea 1 de la Dirección

— Línea 2 de la Dirección

— Línea 3 de la Dirección

— Nombre del 1.er artículo, cantidad y precio unitario.

— Nombre del 2.º artículo, cantidad y precio unitario.

— Nombre del 3.er artículo, cantidad y precio unitario.

— Impuesto a aplicar.

2) *Fase de cálculo:* En una primera parte se escribirán la fecha y la dirección.

En una segunda parte se imprimirá el nombre del artículo, la cantidad, y el precio unitario y el importe. Finalmente este importe se acumulará en la variable que significa total. Esto se repetirá tres veces para cada línea de artículo.

Finalmente se imprimirá el total, se calculará el valor del impuesto y se imprimirá el total correspondiente. Este valor total se imprimirá.

En nuestro ejemplo, la fase de cálculo y de emisión de resultados es simultánea, pero no en todos los programas sucede así.

El principal problema a resolver ahora es cómo encolumnar los números. En este programa hay que repetir muchas veces esto del encolumnamiento. Por eso, vamos a estudiar un caso con cierto detalle para aplicar el procedimiento. Se trata de escribir un número de 4 columnas, que obviamente será menor de 9999, de tal manera que las cifras de las unidades queden en el posición más a la derecha. Para ello, transformamos el número a texto mediante la función STR$, y averiguaremos su longitud; lo haremos mediante una variable intermedia (a$) y mediante la función LEN, a la que restamos un 1. En otra variable (que llamaremos b$) colocaremos 4 espacios en blanco y mediante una asignación colocaremos el texto (a$) en su parte derecha. El esquema de este proceso sería el siguiente:

```
LET a$ = STR$ (q)
LET l = LEN (a$) – 1
LET b$ = LEN (a$)
LET b$ = (4 – l TO 4) = a$
```

Hagamos un comentario a la última instrucción que hemos escrito: En ella se asigna desde la posición 4 – l hasta la 4 con el contenido de a$. Para ver el efecto que tiene esta instrucción veamos la tabla que resulta ante la posibilidad de que a$ sea un número de 1, 2, 3 ó 4 cifras:

a$	l		*El resultado será*
1	0	b$ (4 TO 4) = a$	ƀƀƀ1
12	1	b$ (3 TO 4) = a$	ƀƀ12
123	2	b$ (2 TO 4) = a$	ƀ123
1234	3	b$ (1 TO 4) = a$	1234

Después de todas estas observaciones y notas que, aunque sabemos que son algo pesadas le meten de lleno en la programación, pasemos a la escritura del programa.

Paso 3: Escritura del programa

Ahora teclee el texto del programa de la figura 1 con mucha atención y procurando reconocer en el mismo cada uno de los comentarios que hemos venido haciendo. El programa es largo y, por tanto, tómese el tiempo suficiente y con la tranquilidad necesaria. Si lo hace deprisa y no repasa adecuadamente cada instrucción, el programa no funcionará. Intente comprender el porqué de cada instrucción. Muchas instrucciones serán prácticamente iguales que otras ya tecleadas anteriormente. Le recomendamos en estos casos utilizar la tecla EDIT, con lo que sólo tendrá que variar el número de línea y algún otro carácter en algún caso, pero le liberará de tener que repetir toda la línea. Si tiene dudas de cómo hacerlo, repase el capítulo 2 de «Prácticas con el microordenador». Pero, ojo, no vaya a borrar alguna línea escrita y mucha atención a los espacios en blanco entre comillas; que sean los señalados.

Si ya ha escrito el programa de la figura 1 de este capítulo siga los comentarios que haremos de algunas de las instrucciones. Es más, con lápiz vaya separando los bloques que comentamos sobre el programa impreso en la figura:

— Las sentencias 10 a la 140 son entradas de datos, que tienen un reflejo en la pantalla mediante la instrucción PRINT.

— Las instrucciones 500 hasta la 541 están dedicadas a la impresión de la fecha y la dirección. La instrucción 501 se ha colocado únicamente para que usted pueda observar fácilmente el encolumnado de cada elemento de la factura.

— La instrucción 560 coloca el total acumulado a cero.

— Los bloques 561 a 730, 740 a 920 y 930 a 1120 son bloques prácticamente idénticos, y corresponden a la escritura y cálculo de la línea asociada a un artículo.

```
 10 INPUT "Fecha:";f$ : PRINT "Fecha:";f$
 20 INPUT "Dir 1:";d$ : PRINT "Dir 1:";d$
 30 INPUT "Dir 2:";e$ : PRINT "Dir 2:";e$
 40 INPUT "Dir 3:";g$ : PRINT "Dir 3:";g$
 50 INPUT "Ar 1:";o$  : PRINT "Ar 1:";o$
 60 INPUT "Cantidad:";q1 : PRINT "Cantidad:";q1
 70 INPUT "Precio Unitario:";p1 : PRINT
         "Precio unitario:";p1
 80 INPUT "Ar 2:";p$  : PRINT "Ar 2:";p$
 90 INPUT "Cantidad:";q2 : PRINT"Cantidad:";q2
100 INPUT "Precio Unitario:";p2 : PRINT
         "Precio unitario:";p2
```

Figura 1: Programa para hacer una factura.

```
110 INPUT "Ar 3:";q$ : PRINT "Ar 3:";q$
120 INPUT "Cantidad:";q3 :
          PRINT "Cantidad:";q3
130 INPUT "Precio Unitario:";p3 :
          PRINT "Precio unitario:";p3
140 INPUT "Impuesto:";tpc : PRINT "Impuesto:"
          ;tpc
500 CLS
501 PRINT "12345678901234567890123456789012"
510 PRINT TAB 9;f$ : PRINT : PRINT
520 PRINT TAB 15;d$
530 PRINT TAB 15;e$
540 PRINT TAB 15;g$
541 PRINT : PRINT : PRINT
550 REM Impresion de los articulos
560 LET T=0
561 LET l = LEN o$ : LET b$="..............."
562 LET b$ ( 1 TO l ) = o$
570 PRINT b$;" ";
580 LET a$ = STR$(q1)
590 LET l = LEN a$ -1
600 LET b$ = "    "
610 LET b$(4-l TO 4)=a$
620 PRINT b$;" ";
630 LET a$ = STR$(p1)
640 LET l = LEN a$ -1
650 LET b$ = "    "
660 LET b$(4-l TO 4)=a$
670 PRINT b$;" ";
680 LET  i=p1*q1 :LET t = t+i
690 LET a$ = STR$(i)
700 LET l = LEN a$ -1
710 LET b$ = "     "
720 LET b$(5-l TO 5)=a$
730 PRINT b$;" "
740 LET l = LEN p$ : LET b$="..............."
750 LET b$ ( 1 TO l ) = p$
760 PRINT b$;" ";
770 LET a$ = STR$(q2)
780 LET l = LEN a$ -1
790 LET b$ = "    "
800 LET b$(4-l TO 4)=a$
810 PRINT b$;" ";
820 LET a$ = STR$(p2)
830 LET l = LEN a$ -1
840 LET b$ = "    "
850 LET b$(4-l TO 4)=a$
860 PRINT b$;" ";
870 LET  i=p2*q2 :LET t = t+i
880 LET a$ = STR$(i)
```

```
 890 LET l = LEN a$ -1
 900 LET b$ = "     "
 910 LET b$(5-l TO 5)=a$
 920 PRINT b$;" "
 930 LET l = LEN q$ : LET b$="..............."
 940 LET b$ ( 1 TO l ) = q$
 950 PRINT b$;" ";
 960 LET a$ = STR$(q3)
 970 LET l = LEN a$ -1
 980 LET b$ = "    "
 990 LET b$(4-l TO 4)=a$
1000 PRINT b$;" ";
1010 LET a$ = STR$(p3)
1020 LET l = LEN a$ -1
1030 LET b$ = "    "
1040 LET b$(4-l TO 4)=a$
1050 PRINT b$;" ";
1060 LET  i=p3*q3 :LET t = t+i
1070 LET a$ = STR$(i)
1080 LET l = LEN a$ -1
1090 LET b$ = "     "
1100 LET b$(5-l TO 5)=a$
1120 PRINT b$;" "
1500 REM  escritura de totales
1501 PRINT TAB 26;"-----"
1510 PRINT TAB 19;"Total..";
1520 LET a$ = STR$(t )
1530 LET l = LEN a$ -1
1540 LET b$ = "     "
1550 LET b$(5-l TO 5)=a$
1560 PRINT b$;" "
1570 PRINT : LET i = INT(t*tpc/100+0.5)
1580 PRINT TAB 15;"Impuesto ";STR$(tpc);" ";
1590 LET a$ = STR$(i)
1600 LET l = LEN a$ -1
1610 LET b$ = "     "
1620 LET b$(5-l TO 5)=a$
1630 PRINT b$;" "
1640 PRINT TAB 26;"-----"
1650 LET t = t+i
1660 LET a$ = STR$(t)
1670 LET l = LEN a$ -1
1680 LET b$ = "     "
1690 LET b$(5-l TO 5)=a$
1700 PRINT TAB 26;b$;" "
1710 PRINT TAB 26;"====="
5000 REM Encadenamiento
5010 REM Para hacer otra factura
5020 INPUT "Espero",a$
5030 GO TO 10
```

Dentro de cada uno de estos bloques (comentaremos únicamente el primero) el proceso se divide en cuatro partes:

1. Comprende las instrucciones 501 a 570. Se calcula el nombre del artículo con puntos al final hasta completar 15 columnas. En l (ele) se coloca la longitud del nombre del artículo, se prepara la variable b$ con 15 puntos y se asignan los l (ele) primeros lugares a las variables b$ con el nombre del artículo. Finalmente se imprime.

2. Comprende las instrucciones 580 a 620. Se realiza aquí el mecanismo descrito para colocar el número encolumnado por la derecha.

3. Comprende las instrucciones 630 a 670. La instrucción 680 calcula el importe del artículo y lo añade al total acumulado. Las instrucciones 690 a 730 son el mecanismo para encolumnar un número por la derecha y en este caso concreto de 5 columnas.

Debe observar que cada una de estas partes termina con un PRINT y éste a su vez con un punto y coma (;) excepto en el caso último. De esta manera se va calculando e imprimiendo cada elemento en el interior de la línea.

4. Otro bloque de instrucciones lo constituye la escritura de totales, que corresponde a las instrucciones 1500 a 1710. Se utiliza también frecuentemente la técnica de ajustar a la derecha los números, instrucciones 1520 a 1560, 1590 a 1620 y 1660 a 1690. Cabe resaltar la instrucción 1570, que calcula el valor del impuesto redondeando siempre a unidades enteras. (Si no recuerda esto, repase las funciones numéricas en el capítulo anterior.)

Las instrucciones 5000 a 5030 sirven para detener el programa después de realizar la factura y no cambiar la pantalla para nada. La instrucción 5020 es la instrucción INPUT, que cumple la misión de esperar hasta que no le entremos alguna cosa.

Paso 4: Pruebas

Para comprobar el programa debemos realizar un conjunto de pruebas que nos muestren el funcionamiento del mismo. Incluso es conveniente tenerlas preparadas antes de empezar a teclear, pues a medida que se va entrando el programa es posible probar trozos del mismo. Por ejemplo, en el programa propuesto una vez entrada la instrucción 140, puede ejecutarlo mediante la instrucción RUN y observará el funcionamiento de este trozo de programa.

Después podría haber tecleado hasta la instrucción 550 y el programa continuaría con un GOTO 500 tecleado de modo inmediato con lo cual se ejecutaría las instrucciones 500 y siguientes.

Esto puede hacerlo con cada bloque e ir comprobando poco a poco el funcionamiento del programa. Le recomendamos que utilice el GOTO 500 muchas veces, pues así se evita la entrada de datos.

Le sugerimos ahora que ejecute el programa haciendo las pruebas siguientes y con los datos que se dan:

1. Fecha: 1/5/85

   Dir. 1: Colegio MGC

   Dir. 2: c/ Artistas, 27

   Dir. 3: 08026 Barcelona

Art. 1: Folios	3	150
Art. 2: Lápices	5	20
Art. 3: Gomas	10	5

   Impuesto: IVA 6 %

2. Teclear exactamente los textos de la figura 1 de este capítulo.

   1. Observe que le aparecerá un error:

   3 Subscript wrong, 562: 1

que le indica que hay un índice erróneo en la primera instrucción de la línea 562. Este error es debido a que «Paquetes de 100 folios» supera los 15 espacios reservados para el nombre del artículo.

3. Vuelva ahora a la primera prueba, pero teclee las cantidades con números fraccionarios. Por ejemplo:

Art. 1: Folios	3	12.8
Art. 2: Lápices	5	1.25
Art. 3: Gomas	10	5

Observe que los números calculados son correctos, pero el encolumnado se ha perdido debido a los decimales. Observe también que el impuesto da un número sin decimales, debido al redondeo.

4. Haga la misma prueba que en el caso anterior, pero utilice para el precio de los folios 125.7. Aparece el mensaje de error «3 Subscript wrong, 660: 1», que indica que el subíndice es incorrecto en la primera instrucción de la sentencia 660. Ello se debe a que 125.7 contiene 5 símbolos y el campo está reservado únicamente para 4. En la instrucción 660 hacemos:

```
b$ (−l TO 4) = a$
```

pues l (ele) es igual a 5.

En resumen, que todas estas cuestiones deben tenerse en cuenta y debemos incorporarlas en la lista de restricciones de nuestro programa.

Desde el punto de vista profesional debemos protegernos siempre contra datos que el programa manipule mal.

Queremos dar por concluida esta primera práctica, con la recomendación de que un análisis como el que hemos venido haciendo, aunque sea un poco lento, será un medio muy eficaz para comprender el qué y el porqué

de cada una de las sentencias del programa. Al mismo tiempo será la mejor ayuda para irnos mentalizando nosotros mismos en la forma de elaborar un programa.

4.3.2 Segunda práctica

Paso 1: Definición del problema

El programa que se propone es para calcular el cambio de divisas. Por consiguiente se debe entrar cuánto cuestan en nuestra unidad monetaria un dólar USA, un marco alemán y un yen japonés. El programa debe contestar cuántos dólares, cuántos marcos o cuántos yens nos darán por una determinada cantidad de nuestras unidades monetarias.

Paso 2: Datos

Los datos para el programa son:

a) Cambio del dólar USA: d

b) Cambio del marco alemán: m

c) Cambio del yen japonés: y

d) Cantidad de unidades monetarias que deseamos cambiar: c

El programa constará de:

— Entrada de la tabla de conversión y luego realizaremos un bucle que nos irá pidiendo la cantidad a invertir.

— El cálculo a realizar, que será simplemente dividir la cantidad de nuestras monedas monetarias entradas por el valor del cambio de las respectivas monedas, y seguidamente imprimir.

Paso 3: Escritura del programa

En la figura 2 se presenta el listado del programa. En él se pueden distinguir 4 partes:

— De las instrucciones 10 a la 30, en la que simplemente preparamos los textos d$, m$ y y$ para escribir los resultados. Suele ser una buena costumbre el agrupar al principio del programa los textos fijos.

— En las instrucciones 100 a 120 donde se entran los valores de las monedas respecto a nuestra moneda

— En las instrucciones 200 a 450 se entran las unidades monetarias que deseamos cambiar y se imprime la tabla. En este punto se utilizan las variables que se han preparado en las sentencias de la 10 a la 30.

```
  10 LET d$=" dolares USA"
  20 LET m$=" marcos alemanes"
  30 LET y$=" yens"
 100 INPUT "Dolar USA:";d
 110 INPUT "Marco aleman:";m
 120 INPUT "Yen japones";y
 200 INPUT "Cantidad:";c
 400 REM Fase de calculo
 410 PRINT c;" unidades monetarias son:"
 420 PRINT TAB 5;c/d;d$
 430 PRINT TAB 5;c/m;m$
 440 PRINT TAB 5;c/y;y$
 450 PRINT
1000 GO TO 200
```

Figura 2: Programa para calcular el cambio de divisas.

— La instrucción 1000 recicla el cálculo al pedirnos una nueva cantidad.

Paso 4: Pruebas

En este paso dejamos que sea usted mismo el que realice las pruebas del programa.

4.3.3 Tercera práctica

Paso 1: Definición del problema

Se trata de construir un programa para conocer el coste de un plato determinado en un restaurante, conociendo la composición y los precios del mercado en un día determinado.

Se supone que el plato tiene una composición fija, que es la siguiente:

200 gr de calabacín

80 gr de espinacas

25 gr de mantequilla

Para facilitar el programa, la composición del plato suponemos que sólo consta de estos componentes.

Paso 2: Datos

En una primera fase definiremos las variables de composición en lo que se refiere a unidades y datos:

c1 = 200 a$ = «gr de calabacín»

c2 = 80 b$ = «gr de espinacas»

c3 = 25 c$ = «gr de mantequilla»

A continuación se entran los precios de cada uno de estos componentes y procederemos al cálculo.

Habrá además una variable que sea el total acumulado.

Paso 3: Escritura del programa

En la figura 3 le presentamos el programa para resolver este problema. Sin embargo, antes de mirarlo, intente escribirlo usted. No se preocupe si no coincide exactamente con el que le proponemos. No significa por ello que esté mal. El único criterio importante es que ejecute lo que se le pide.

De todas maneras, si no llega a conseguirlo no se preocupe y en todo caso estudie y pruebe nuestra versión comparándola con lo que usted haya hecho o intentado hacer.

```
  10 LET c1=200: LET a$=" grs. calabacin"
  20 LET c2=80:  LET b$=" grs. espinacas"
  30 LET c3=25 : LET c$=" grs. mantequilla"
 100 INPUT "Precio calabacin:";p1
 110 INPUT "Precio espinacas:";p2
 120 INPUT "Precio mantequilla:";p3
 500 CLS : LET t = 0
 510 LET v= p1*c1/1000. : LET t=t+v
 520 PRINT c1;a$;" a ";p1;" = ";v
 530 LET v= p2*c2/1000. : LET t=t+v
 540 PRINT c2;b$;" a ";p2;" = ";v
 550 LET v= p3*c3/1000. : LET t=t+v
 560 PRINT c3;c$;" a ";p3;" = ";v
 570 PRINT "Total = "; t
1000 GO TO 100
```

Figura 3: Programa para calcular el coste de un plato.

Índice

Capítulo 4. Operadores relacionados

PARTE II. PRACTICAS CON EL ORDENADOR

Capítulo 1

Capítulo 2

Capítulo 3

Capítulo 4

ENCICLOPEDIA DEL BASIC

SPECTRUM

1

- Programación de ordenadores
- Arquitectura del microordenador
- El software
- Primeras instrucciones del BASIC
- Los programas BASIC
- Las instrucciones REM y PRINT
- Las instrucciones TAB e INPUT
- Las instrucciones STOP y GOTO
- La instrucción LET
- Expresiones aritméticas
- Representación interna de datos
- Expresiones textuales
- Funciones intrínsecas
- Funciones numéricas
- Funciones textuales
- Funciones numéricas y argumento textual
- Funciones textuales y argumento numérico
- Control de pantalla
- Operadores relacionales
- Igualdad y desigualdad
- Comparaciones entre conjuntos ordenados
- La codificación
- Instalación del ZX Spectrum y ZX Spectrum +
- El teclado del Spectrum
- Comandos de instrucciones

ceac